Badges-io: Gamification com a exemple de microservei

President: Enric Mayol Sarroca

Vocal: Carlos Seara Ojea

Tutora: Carme Quer Bosor

Estudiant: Miquel Mariño Espinosa

1. MOTIVACIÓ DEL PROJECTE

1.1. Problema inicial

Atrápalo S.L. [1] és una empresa creada l'any 2000 amb 15 anys d'antiguitat dedicada a la venta online d'oci urbà i oci de vacances [2]. L'oferta d'oci urbà és,entre d'altres, entrades per a concerts de música, entrades per obres de teatre, així com reserves per a restaurants. L'oferta d'oci de vacances esta focalitzada, bàsicament, en venta de viatges i estances en hotels.

En els últims anys, l'empresa ha dedicat molts recursos en millorar l'experiència dels usuaris que utilitzaven la web, basant-se en la certesa de que un usuari satisfet és un potencial client habitual.

Un cas d'èxit per a millorar l'experiència d'usuari ha sigut migrar la aplicació a responsive web design [3]. Aquest tipus de disseny permet l'adaptació als diferents tipus de dispositius (mòbil, tablet, escriptori) els quals són emprats pels usuaris per poder accedir a la pàgina web sense tenir la necessitat de crear una solució software per a cada un d'ells, unificant d'aquesta forma, estils, colors e interaccions del portal web. El resultat final del projecte va ser una **millora molt gran en la sensació de confort** durant la navegació per part l'usuari de la web, materialitzant-se en una millora sensible en la conversió de venta de productes. Es a dir, **es va incrementar de forma significativa el percentatge de persones que visitaven la web i acabaven realitzat una compra**.

Degut als bons resultats obtinguts en els darrers projectes, la direcció de l'empresa ha decidit que vol continuar apostant per **millorar l'experiència d'usuari**.

Una idea que ha sorgit és poder **donar als usuaris una oferta més enfocada al seu perfil com a consumidors**. En aquest escenari, l'usuari hauria d'escollir en un inventari de productes molt més adequat a les seves necessitats, augmentant considerablement la probabilitat de que l'usuari acabi realitzant una compra, millorant d'aquesta forma encara més el **rati actual de conversió d'usuaris que finalitzen una compra**.

La web actualment ja incorpora algunes eines orientades a adequar l'oferta de producte a les necessitats de l'usuari. Per exemple, existeixen

eines les quals consisteix en enviar correus electrònics de forma periòdica d'ofertes a usuaris que hagin comprat alguna vegada. En funció de l'històric de les compres que ha realitzat l'usuari s'adapta l'inventari dels productes i de les ofertes que s'inclouen en el correu electrònic que se l'hi envia. Tot hi ser una molt bona aproximació, l'inconvenient d'aquestes eines és que no gestionen perfils d'usuaris, sinó que envia un correu personalitzat per a cada usuari. Si es creessin perfils d'usuaris es podrien afegir campanyes més enfocades als diferents perfils d'usuaris que es poguessin identificar, donant una imatge més social de l'empresa. L'existencia de perfils podria també millorar l'experència dels usuaris registrats a miatrapalo ja que un cop un usuari s'hagués autenticat en la web podria veure ofertes creades en funció del seu perfil.

Per aquestes raons i en el context actual, **la idea proposada per la direcció de l'empresa crea la necessitat de gestionar perfils d'usuari** per a poder enfocar l'oferta de producte a les seves preferències com a consumidor.

1.2. Necessitats

Per a poder dur a terme la idea proposada, és necessari poder **gestionar perfils d'usuari**. L'objectiu de la creació i gestió d'aquest perfils és poder identificar els usuaris del sistema en grups els quals comparteixen preferències i necessitats similars. Malauradament, aquesta funcionalitat no es troba actualment disponible en el portal web. D'aquesta forma, s'identifica la necessitat d'implementar mitjançant algun tipus de tecnologia un sistema que permeti gestionar perfils d'usuari d'una forma eficient i intuïtiva.

1.3. Objectiu

Per a poder satisfer la necessitat creada i així implementar amb èxit la idea proposada, ***l'objectiu del projecte serà el de crear una eina que permeti gestionar perfils d'usuari***. Aquesta eina ajudarà a millorar l'experiència de navegació a través de la web per part de l'usuari, reforçant la seva fidelització i retenció al sistema. Finalment amb aquesta millora, es preveu que es traduirà en un increment del rati de conversió d'usuaris que acaben per realitzar una compra en el portal web.

2. GAMIFICATION

2.1. Concepte

S'enten per **Gamification** [4] l'ús de tècniques, elements i dinàmiques pròpies dels jocs i l'oci, aplicades a activitats no recreatives amb l'objectiu de potenciar la motivació, així com de reforçar la conducta per a solucionar un problema o per aconseguir una meta prefixada.

Figura 1 2.1 Gamification com a tècnica motivació

La introducció de tècniques provinents dels jocs permet **transformar una activitat a priori sense al·licient en una activitat més interessant per l'usuari**. Una de les tècniques més aplicades és la recompensa a l'usuari que participa en l'activitat pel fet d'aconseguir un objectiu plantejat. Les recompenses solen ser **l'assignació d'insígnies** o, com s'anomena aquest element dins del concepte de gamification, **badges**. Aquest fet és en essència un reforç positiu el qual té com a objectiu augmentar la motivació de l'usuari per a que continuï participant en l'activitat [5].

En la vida quotidiana trobem un clar i senzill **exemple** d'aplicació d'aquest concepte: les cadenes de **supermercats**. En la majoria de supermercats, quan un client realitza una compra pot acumular punts amb la seva targeta de client en proporció a la quantitat total de diners que s'ha gastat en l'establiment. Aquests punts poden ser canviats per productes o altres tipus de beneficis que ofereix el supermercat. D'aquesta forma es realitza una estratègia de fidelització de clients utilitzant els punts acumulats en les compres com a badges. Els badges es converteixen en un reforç positiu per a l'usuari de l'establiment per a que continuï consumint els seus productes.

En el context del **<u>món empresarial internacional</u>**, els pioners en aplicar aquest concepte van ser **<u>American Airlines</u>** [6], l'any 1981. Es tracta d'una aerolínia amb seu ubicada a Fort Worth, Texas (Estats Units). Els responsables de l'àrea de comerç d'aquesta empresa tenien la necessitat en aquell moment de generar més vendes i se'ls va acudir crear programes de fidelització d'usuaris. Un d'aquests programes és el de ***<u>viatgers freqüents</u>*** [7][8] també anomenat AAdvantage. Aquest programa ofereix als seus viatgers acumular milles dels viatges realitzats, les quals poden ser intercanviades per beneficis tals com altres viatges, lloguers de cotxes, estances en hotels o descomptes en botigues. Les <u>milles acumulades en aquest context serien els badges</u> en l'ambit de la gamification. D'aquesta manera, l'empresa recompensa als seus usuaris més habituals amb avantatges especials, reforçant la seva fidelització com a clients de l'empresa. En els següents anys la resta d'aerolínies van seguir l'exemple d'American Airlines creant les seves pròpies avantatges per a viatgers freqüents. Es calcula que actualment, des del llançament de la oferta, un total de 14 trilions de milles de viatger freqüent han sigut acumulades per persones arreu del món, el que suposa en dòlars americans un total de 700 bilions.

Figura 2 2.1 Logo American Airlines

Figura 3 2.1 Viatger Freqüent

Continuant amb els exemples d'aplicació de gamification, **<u>Stackoverflow</u>** [9] n'és un altre en el context, en aquest cas, d'una **comunitat online d'usuaris**. Es tracta d'un portal web creat per Jeff Attwood y Joel Spolsky a l'any 2008 el qual és utilitzat per una comunitat de

desenvolupadors informàtics on es poden trobar solucions a problemes i dubtes de desenvolupament del software en diferents tipus de llenguatges de programació [10]. El seu funcionament consisteix, bàsicament, en què un usuari realitza una pregunta i la resta d'usuaris la contesten. Durant aquest procés s'avaluen les respostes mitjançant un sistema de vots en el qual poden participar tots els usuaris amb l'objectiu d'identificar quina és la millor resposta. L'aplicació del concepte de gamification en aquest context consisteix en definir diferents perfils d'usuaris i de badges en funció de paràmetres [11] com poden ser el tipus de preguntes que realitzen, el tipus de respostes que donen o el tipus de participació que fan en un determinat període de temps. Així, per exemple, podem trobar un badge de tipus Stellar Question [12] el qual s'assigna a l'usuari que ha realitzat una pregunta que ha estat escollida com a favorita per un total de 100 usuaris o el badge Guru [13] que s'assigna a un usuari amb una resposta que tingui 40 o més vots a favor. Aquest cas d'aplicació de gamification, des d'un punt de vista social, és molt interessant, pel fet de que ***els badges que s'assignen als usuaris classifiquen, a la vegada, els usuaris en perfils o rols que els caracteritzen dins de la pròpia comunitat***.

Figura 4 2.1 Logo StackOverflow

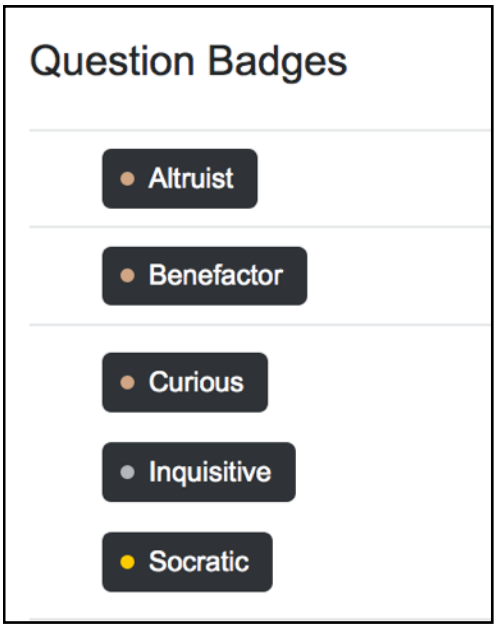


Figura 5 2.1 Badges StackOverflow

Finalment, l'aplicació **Foursquare** [14] n'és un altre exemple digne de ser citat en aquest apartat. Es tracta d'un servei basat en geolocalització web aplicada a les xarxes socials que permet localitzar un dispositiu fixe o

mòbil en una ubicació geogràfica determinada. Aquest servei va ser creat a l'any 2009 per Dennis Crowley i Naveen Selvadurai [15]. L'aplicació és molt utilitzada per a poder trobar i recomanar llocs d'oci urbà tal com poden ser restaurants, sales de concerts o bars. L'usuari un cop ubicat en un lloc, realitza l'acció de check-in per a geolocalitzar el lloc i el valora realitzant votacions i comentaris. D'aquesta forma es crea una comunitat de persones que fan recomanacions de llocs interessants i els propietaris d'aquests llocs es beneficien indirectament de l'activitat dels usuaris de l'aplicació gràcies a la publicitat que aquests les hi fan dels seu negocis. En aquesta aplicació, existeixen **diferents tipus de badges** [16], tal com el badge Explorer que se li assigna a aquell usuari que ha realitzat 25 check-ins d'ubicacions diferents o el badge Local que s'aconsegueix quan es realitza 3 check-ins en la mateixa ubicació en una setmana. Semblant al cas de stackoverflow, ***l'asignació de badges crea perfils o rols dins de la comunitat d'usuaris de l'aplicació***.

Figura 6 2.1 Logo Foursquare

Figura 7 2.1 Badges Explorer i Local

2.2. Context del projecte

Després de l'estudi realitzat en l'apartat anterior sobre el concepte de gamification i analitzant els exemples citats on s'aplica el concepte en el context del món empresarial, en aquest apartat es justifica perquè l'aplicació del **concepte de gamification satisfà les necessitats plantejades** i es descriu com es pot aplicar aquest concepte en el context de l'empresa Atrapalo.

L'objectiu de millorar l'experiència de l'usuari durant la navegació, per a reforçar la seva fidelització com a client i millorar, d'aquesta forma, el rati de conversió de compres realitzades per part dels usuaris, fa necessari

crear una **eina per a poder gestionar perfils o rols** per a que es pugui enfocar l'inventari de productes que ofereix la web a les necessitats dels usuaris com a consumidors. Un inventari més enfocat de productes a les preferències del client augmentaria de forma considerable la probabilitat de que l'usuari faci una compra.

Analitzant el cas de l'empresa American Airlines i la seva estratègia de fidelització d'usuaris viatgers freqüents, a priori sembla raonable que el concepte pugui aplicar-se en el context d'Atrapalo. Com ja s'ha comentat a l'inici de la memòria, Atrapalo ofereix, d'entre altres productes del seu extens inventari, viatges en avió i es podria aplicar la mateixa estratègia de fidelització de clients a aquest producte. Aplicant aquesta estratègia es podria satisfer la necessitat de poder enfocar el producte a les preferències de l'usuari com a consumidor en el cas particular del producte de viatges en avió, així com oferir noves ofertes dirigides a aquest tipus d'usuaris en particular. No obstant, la diferencia entre American Airlines i Atrápalo és que la segona té una oferta que no només es limita als viatges en avió, sinó que també es venen altres productes com poden ser entrades pel teatre o reserves en restaurants. Encara que en una primera instància podria semblar lògic aplicar la mateixa estratègia en els diferents productes amb diferents noms, com per exemple el comensal freqüent en el producte de reserves de restaurants, des del departament comercial de l'empresa alerten de que pot existir el risc d'homogeneïtzar els diferents programes de fidelització per a tots els productes i perdre la flexibilitat de tractar a cada producte d'una forma individualitzada i autònoma.

Els exemples de les empreses Stackoverflow i Foursquare que apliquen els concepte de badges i els assignen en funció d'uns paràmetres basats en la interacció que realitza l'usuari amb el sistema és una aproximació molt interessant. Aquests badges tenen noms molt semàntics (Guru, Explorer, etc.) amb un definició pròpia dins dels contextos de les aplicacions on de forma indirecta es creen perfils i rols amb els que la comunitat identifica als usuaris. Aplicar aquesta estratègia d'assignació de badges satisfà la necessitat inicial del projecte de crear i gestionar perfils, complint amb la funció de reforç positiu de cara a l'usuari, incrementant la seva fidelització al sistema. No obstant, en cap dels dos casos sembla que

tinguin la necessitat de gestionar una oferta de producte en les seves respectives aplicacions, sinó que tan sols s'incentiva a l'usuari a continuar participant en la interacció amb l'aplicació. En aquest context els badges assignats són merament informatius però no influeixen en el comportament i interacció del sistema cap a l'usuari. S'ha de notar que un usuari pot tenir assignats més d'un badge.

Finalment es pot concloure que es tracta d'una aproximació adequada en el context del projecte i que satisfà les necessitats creades utilitzar el **concepte de gamification d'assignació de badges com realitzen les empreses Stackoverflow i Foursquare per a crear perfils i rols d'usuaris i poder enfocar el producte d'una forma acurada i específica en funció dels diferents perfils com realitza d'una forma més genèrica amb el programa de viatger freqüent American Airlines**. En el cas d'Atrápalo per exemple, es podria crear un badge de tipus Melòman per a usuaris que hagin comprat 10 entrades per a concerts o Shakespearians per a usuaris que hagin comprat entrades pel teatre 4 vegades en el període d'un mes. Als usuaris Melòmans se'ls podria aplicar un descompte del 10% en el pròxim concert i oferir-los més concerts del catàleg de productes, ja que sembla que existeix una alta probabilitat de que aquest client compri més entrades per concerts en el futur. Pel cas dels Shakespearians es podria realitzar una oferta similar però enfocada al producte d'entrades de teatre. Com en el cas de Stackoverflow i Foursaquare, es podria assignar més d'un badge per usuari.

Figura 1 2.2 Logo Atrapalo

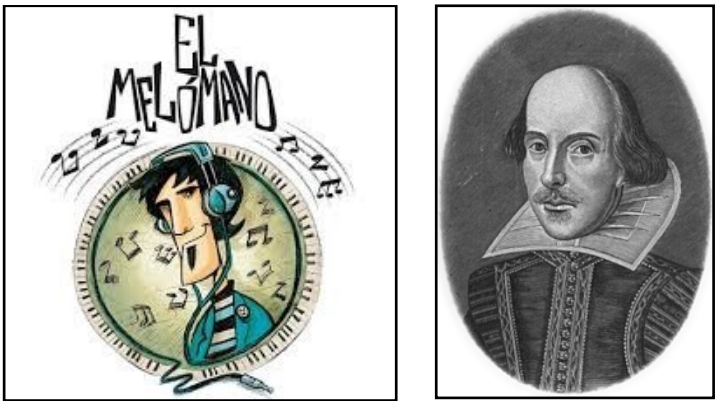


Figura 2 2.2 Badges Meloman i Shakespearia

2.3. Estudi de mercat

Una vegada presa la decisió d'aplicar el concepte de gamification, es realitza un estudi de mercat per analitzar si existeixen solucions implementades les quals permetin gestionar la creació i assignació de badges així com l'impacte de la seva integració al sistema actual.

MoodleBadges [17] és una llibreria de codi lliure creada per l'equip de Learnbrite. Ofereix diferents tipus de badges que es poden fer servir integrant la llibreria al sistema. Sembla ser que aquesta llibreria tan sols ofereix els badges per defecte que porta la llibreria. La seva documentació és escassa pel que suposaria un gran esforç la seva integració al sistema actual així com el seu manteniment manteniment.

Pels portals web creats amb la tecnologia wordpress existeix el plugin **YITH WooCommerce Badge Management** [18] per a poder gestionar i assignar badges a una aplicació. En la descripció del pluguin, els seus creadors asseguren que utilitzar gamification pot fer incrementar les ventes un 55% d'una aplicació. La integració al sistema seria costosa ja que la web d'Atrapalo no esta creada amb wordpress.

Per últim, **OpenBadges** [19] és una eina open source creada per l'empresa Mozilla. Aquesta eina no només permet crear i gestionar badges sinó que a més permet la seva difusió a través de la xarxa entre els usuaris. D'aquesta forma, els usuaris poden utilitzar badges que uns altres usuaris hagin creat. Es pot aplicar a qualsevol tipus de model de negoci o d'aprenentatge. L'impacte de la integració amb el sistema actual no seria elevat pel fet que es tracta d'una eina open source totalment configurable i adaptable. Tot i que sembla una eina molt adequada en el context del projecte, la direcció de l'empresa ha decidit dedicar recursos per crear una eina pròpia, ja que disposa d'un equip tècnic d'alta qualitat i prefereix encarregar-li la tasca a aquest equip en comptes de confiar en una tecnologia a priori desconeguda.

3. ANÀLISI DE REQUISITS

3.1. Requisits funcionals

En aquest apartat **es descriuen les funcionalitats** que hauria d'oferir el sistema, el que es coneix com a requisits funcionals.

L'eina que es vol dissenyar i implementar hauria de poder **gestionar usuaris de forma bàsica**. Hauria de ser capaç de donar d'alta usuaris i poder consultar les seves dades bàsiques.

Per un altre banda, el sistema hauria de ser capaç de **gestionar badges**. L'eina ha de donar d'alta (create), consultar (read), modificar (update) i esborrar badges (delete). Aquest tipus de sistema de gestió que agrupen totes aquestes funcionalitats es sol anomenar en el disseny del software com a **sistema CRUD**, l'acrònim dels noms de les funcionalitats esmentades.

3.2. Requisits no funcionals

En aquest apartat **es descriuen les característiques desitjables** que hauria de tenir l'eina que es vol implementar, el que es coneix com a requisits no funcionals. També es descriu el tipus de tecnologia necessària per implementar el sistema.

L'**escalabilitat** de l'eina és fonamental. Un sistema escalable per definició facilita la implementació de noves funcionalitats d'una forma senzilla i ràpida. És una característica que esta directament relacionada amb la rendibilitat del producte de tal forma que, **assignant pocs recursos, es poden implementar noves funcionalitats d'una forma eficient i eficaç**.

Una altre de les característiques desitjables és la **mantenibilitat**. Els sistemes que presenten aquesta característica **faciliten la tasca de detecció i correcció de comportaments no desitjats**, el que s'anomenen bugs. Els seu disseny i posterior implementació sol ser intuïtiu facilitant el seu anàlisis i l'aprenentatge del seu funcionament per part dels programadors i enginyers que l'han de mantenir.

L'eina ha de ser a proba d'errors, és a dir, ha de ser **robusta**. Per a poder garantitzar aquesta característica **un sistema ha de poder ser testejat de forma automàtica**. Aquest fet significa que cada vegada que

s'implementi una funcionalitat nova, junt amb la nova funcionalitat cal proveir dels tests necessaris per a que es pugui garantir que funciona adequadament i també poder garantir mitjançant els tests creats amb anterioritat que la resta de funcionalitats segueixen funcionant. D'aquesta manera **es potencia el fet de que l'addició d'una nova funcionalitat no genera comportaments indesitjables a la resta de components** i que el sistema funciona com s'espera que ho faci.

Seria desitjable que l'**eina creada es pugui instal·lar i integrar fàcilment a qualsevol tipus de sistema** o plataforma, ja sigui externa o local. Aquesta caracterísitica s'anomena **portabilitat**. Tan en el cas d'instal·lar l'eina en la infraestructura d'Atrapalo o en la infraestructura d'una altre empresa o en el cas de que un futur canvii la tecnologia de l'empresa, si l'eina és portable, facilitarà la seva adaptació al canvi i la seva integració a la nova infraestructura.

La última característica és la **usabilitat**. L'eina **hauria de ser intuïtiva en el seu ús per part de l'usuari**. Aquest fet fa necessari que es **proveeixi de documentació**, en el millor dels casos online, actualitzada a l'usuari, per a que aquest li pugui treure el màxim profit a les funcionalitats que l'eina ofereix. Un **correcte disseny de la GUI** també es fonamental per a poder potenciar aquesta característica.

Per últim, la **tecnologia emprada** per a poder implementar el sistema es basarà en un **servidor web**. El servidor serà l'encarregat d'oferir les **GUI (Grafic User Interface)** a l'usuari per a que aquest sigui capaç de forma autònoma de gestionar els badges i les seves dades bàsiques com usuari. Seran necessaris **dispositius físics** per a poder emmagatzemar les imatges que s'assignen als badges, així com el emmagatzemar el codi software el qual implementa les funcionalitats del sistema. Aquests dispositius d'emmagatzamatge poden ser tals com **disc durs** localitzats en el servidor web.

4. ESPECIFICACIÓ DEL SISTEMA

El procés d'especificació del software consisteix en identificar les funcionalitats que ha de realitzar el sistema per a modelar-les com a component software. En aquest apartat es descriuen els components necessaris del sistema per a poder representar la realitat. S'identifiquen les entitats de la nova eina que conformaran el nucli del sistema i els casos d´ús els quals defineixen les funcionalitats que oferirà la nova eina.

4.1. Entitats

Per començar el procés d'especificació del software i modelar el sistema s'identifiquen les entitats necessàries per a poder representar la realitat del context del projecte.

Una entitat és el **component bàsic de software que representa una abstracció de la realitat** la qual dins un context adquireix un significat i es pot identificar amb un concepte o objecte real. És la base de la **programació orientada a objectes** on cada objecte s'associa a una entitat. Els objectes són classes que poden estar implementades en qualsevol tipus de tecnologia o llenguatge de programació.

En el context de gamification es poden identificar les següents entitats bàsiques:

- **Badge**

Es tracta del concepte central dins el context de gamification. Els usuaris gestionaran aquestes entitats per poder donar-les d'alta i poder assignar-les quan ho requereixi la situació. Té com atributs de classe el nom (name), descripció (description) i si és compartit o privat (isMultUser). Una badge esta relacionat amb un únic usuari i una única imatge.

- **User**

Són els actors principals. S'encarregaran d'interactuar amb el sistema per a la gestió de la resta d'elements. En el context del projecte, els usuaris són els sistemes externs els quals invocaran les funcionalitats que ofereix la nova eina. Més concretament, l'usuari serà el portal web Atrapalo. Té com atributs de classe l'email (email), el nom d'usuari (username) i el password. Un user esta relacionat amb cap o varis badges.

- Image

En el context del projecte es pot considerar que una imatge d'un badge té prou significat i característiques per si mateixa per a considerar-la com una entitat pròpia. Té com atributs de classe un nom (name), amplada (width), llargada (height) i el format de la imatge (format). Una Imatge estarà relacionada amb un únic Badge.

A continuació, es mostra el diagrama UML que especifica graficament les classes, els seus atributs i les relacions entre elles.

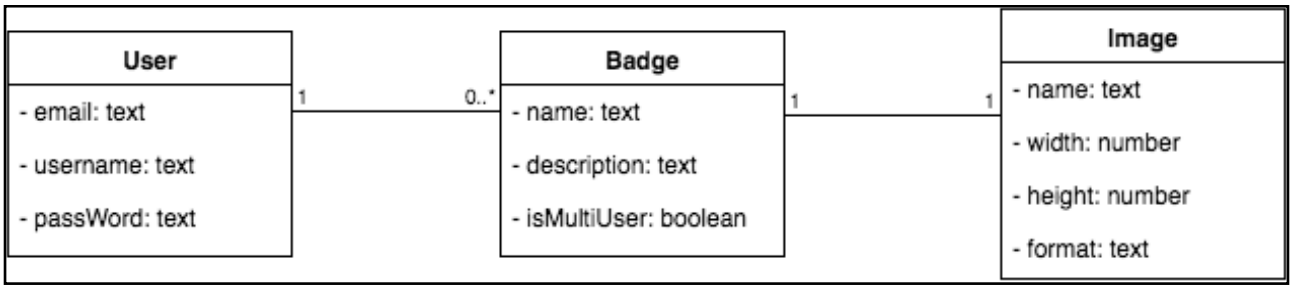


Figura 1 4.1 Diagrama UML de classes

4.2. Casos d'ús

Un cas d'ús és una **descripció dels passos o les activitats que s'hauran de realitzar per poder dur a terme algun procés en concret**. Les **entitats que participen en un cas d'ús són els actors**. En el context d'enginyeria del software, un cas d'ús és una seqüència d'interaccions entre un sistema i els seus actors en resposta a un esdeveniment que inicia un actor sobre el propi sistema.

Els casos d'ús identificaran les funcionalitats que ofereix el sistema. D'aquesta forma, s'han identificat dos tipus de casos d'ús: els que fan referència a la **gestió d'usuaris** i els que fan referència a la **gestió de badges**. A continuació es descriuen els casos d'ús identificats per a la gestió d'usuaris:

- **Sign In**

  Funcionalitat de donar d'alta un usuari en el sistema.

- **Log In**

Funcionalitat d'identificar i donar accés a un usuari.

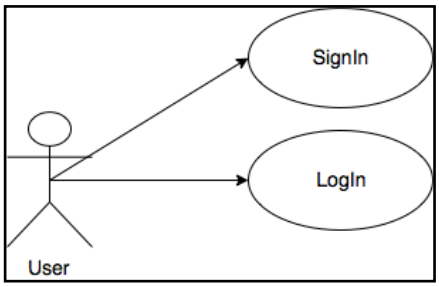


Figura 1 4.2 Casos d'ús usuaris

A continuació es descriuen els casos d'ús identificats per a la gestió de badges:

- **Create Badge**

  Funcionalitat per a crear un badge i associar-li una imatge.

- **Get Badge**

  Funcionalitat per a obtenir un badge donat un identificador.

- **Update Badge**

  Funcionalitat per a modificar dades d'un badge en concret.

- **List Badges**

  Funcionalitat per obtenir una llistat de badges per un usuari.

- **Delete Badge**

  Funcionalitat per esborrar un badge del sistema.

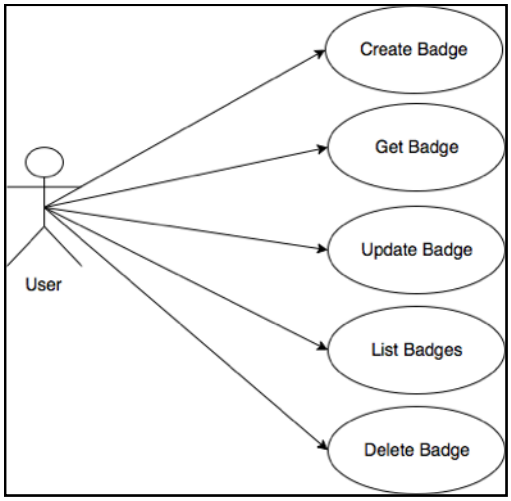


Figura 1 4.2 Casos d'ús gestió badges

5. INTEGRACIÓ AMB EL SISTEMA

Actualment el departament tècnic de l'empresa consta d'uns **90 programadors** en plantilla.

El portal web de Atrapalo està creat amb un **software format per una quantitat molt considerable de fitxers, classes i línies de codi**. Existeix un control de versió de codi que comparteixen tots els productes el qual s'actualitza varies vegades al dia. Aquest repositori és el que allotja el codi més antic, el que es sol anomenar com a legacy o codi heredat. Per a implementar la nova eina **es crearà un repositori nou independent** del repositori que ja existeix actualment. D'aquesta forma, el codi de la nova eina serà totalment independent i desacoblat de la resta de codi del portal web d'Atrapalo.

Pel que fa a la **infraestructura hardware de Atrapalo**, l'empresa compta amb amb **7 servidors frontals** els quals són els encarregats de rebre les peticions i d'entregar contingut que no és dinàmic, com poden ser imatges o contingut html cachejat en sistemes de cache. També consta de **19 servidors web** balancejats on s'allotja el codi software, els quals processen les peticions que generen contingut dinàmic. Finalment, també hi ha **4 màquines on s'allotgen les bases de dades** les quals són compartides per a tots els productes de l'empresa i en les quals s'emmagatzema les dades generades pel sistema.

En els següents apartats es fa un anàlisi del cost de l'impacte de la integració de l'eina en el context del software i hardware d'Atrapalo descrit abans. L'anàlisi de l'impacte es centrarà tant en la integració software com la integració hardware. Es plantegen dos models d'integració tant en un cas com en un altre: integració desacoblada o integració acoblada. Finalment i valorant totes les opcions plantejades, en l'apartat decisió tipus integració, s'indica quina serà l'opció adequada, tenint en compte l'anàlisi previ, de com integrar la nova eina en el sistema actual.

5.1. Integració Software

En aquest apartat s'analitza els avantatges i els inconvenients de la integració de l'eina amb el software actual de l'empresa tant de forma acoblada com de forma desacoblada.

5.1.1. Integració Acoblada

Una integració acoblada al software actual suposa que el codi que implementi **les funcionalitats definides de la nova eina estigui ubicat i mesclat amb el codi preexistent de l'empresa**. Per exemple, si durant el procés de compra es vol gestionar l'assignació d'un badge a un usuari, el codi que s'encarrega d'assignar un badge a l'usuari es trobaria localitzat en el mateix fitxer que el codi que realitza el procés de compra.

En aquesta situació, sembla a priori que el gran ***avantatge*** que oferiria aquest tipus d'integració seria la **facilitat d'integrar l'eina en un software preexistent**. Aquesta facilitat vindria donada pel fet que el codi porta anys funcionant en la infraestructura i en el hardware actual pel que ofereix garanties de que es poden implementar noves funcionalitats amb èxit invertint poc esforç en adaptar o ampliar la infraestructura preexistent. Per una altre banda, el codi legacy ha estat mantingut a diari pels **programadors de l'empresa els quals estan molt familiaritzats en el desenvolupament de noves funcionalitats** pel que els hi seria relativament senzill implementar una nova funcionalitat en el codi actual. En aquest escenari, la velocitat d'integració de l'eina seria molt gran i invertint pocs recursos el projecte es podria finalitzar ràpidament.

Pel que fa als seus ***inconvenients***, el **software preexistent com s'ha comentat en l'apartat anterior fa molts anys que existeix**. Encara que hi hagi parts que s'han modificat i actualitzat, la majoria de components que conformen el nucli de l'aplicació no s'han actualitzat i no hi ha intenció d'invertir recursos actualment per a la seva redefinició i reimplementació. Una altre característica no desitjable que es pot observar en el sistema actual és que **totes les funcionalitats que conformen l'aplicació es troben molt acoblades entre elles**. Aquest fet repercuteix

directament en el manteniment de l'aplicació, dificultant la detecció i correcció d'errors encara que els programadors que s'encarreguin de la seva mantenibilitat estiguin familiaritzats amb el codi. Quan un codi està molt acoblat, ***el fet de modificar una línia d'una funcionalitat té una alta probabilitat d'afectar a una altre funcionalitat diferent, generant un comportament no desitjat del sistema difícil de detectar***. Cada funcionalitat nova que s'integra al sistema de forma acoblada augmenta exponencialment la dificultat del manteniment de tota la aplicació suposant un cost excessiu en el manteniment del sistema.

5.1.2. Integració Software Desacoblada

Una integració desacoblada del software suposa que tot el codi necessari per implementar la nova funcionalitat **es trobi localitzat en una ubicació externa a la localització del codi preexistent**. Una primera aproximació simple seria que el codi es trobi en fitxers i directoris externs del codi preexistent però residint en el mateix control de versions on resideix la resta de l'aplicació. Una altre aproximació més ambiciosa seria implementar tot el codi nou en un control de versions independent i que el sistema faci servir el repositori del codi nou com a dependència. D'aquesta forma el codi podria ser mantingut i ampliat de forma totalment independent al de la resta de l'aplicació.

El gran ***avantatge*** que ofereix aquest tipus d'implementació, sobretot en l'aproximació d'utilitzar un repositori diferent, és l'**aïllament de la nova eina amb la resta del sistema**. D'aquesta forma, **la mantenibilitat** de la nova eina i, per conseqüència, de la resta de l'aplicació, **es facilita**, ja que baixa de forma dràstica la probabilitat que modificant el codi de la nova eina generi errors en la resta de l'aplicació i a la inversa. L'accés del sistema a les funcionalitats que ofereix la nova eina s'haurà de realitzar mitjançant crides a mètodes molt específics i aquest contestarà amb un format pre-acordat de resposta. Tan sols la modificació errònia de la firma dels mètodes (nom i número de paràmetres) o de la resposta d'aquests pot afectar en el comportament de la resta del sistema de forma negativa. Per altre banda, el **codi de la nova eina seria portable**. Pel que la integració en el sistema no

incrementaria el cost de mantenibilitat de la resta del sistema en comparació a realitzar una integració de forma acoblada.

L'***inconvenient*** principal seria que el **temps d'integració de la nova eina augmentaria comparat amb realitzar una integració de forma acoblada**. S'hauria de valorar de quina magnitud estem parlant, però a priori sembla un fet irrefutable que crear un nou repositori suposaria un cost addicional en la integració inicial, així com implementar tot el sistema de comunicació per a que la resta de l'aplicació interactuï amb la nova eina. Pel que suposaria que el projecte seria més en llarg termini i més costós pel que fa a la finalització de la primera entrega de l'eina.

5.2. Integració Hardware

En aquest apartat s'analitza els avantatges i els inconvenients de la integració de l'eina amb el hardware actual de l'empresa tant de forma acoblada com de forma desacoblada.

5.2.1. Integració Acoblada

Una integració acoblada suposaria que el software que implementarà les funcionalitats de la nova eina **residiria en el mateix hardware que el del sistema actual**.

L'***avantatge*** principal d'aquest tipus d'integració seria que els **recursos que s'invertissin per adquirir nou hardware és mínim** ja que, òbviament, es disposa de forma immediata del hardware que utilitza l'aplicació preexistent. Per altre banda, **el hardware actual és i ha estat utilitzat de forma exhaustiva per l'aplicació en un llarg termini de temps**. Aquest ha de servir continguts i interactuar amb milers d'usuaris els quals visiten el portal web de forma diària. Aquest fet garantitza que el hardware actual està dissenyat per a ser robust i d'alta disponibilitat, característica molt desitjable per a que un sistema funcioni de la forma més eficaç i correcte possible.

L'***inconvenient*** de la nova eina de compartir el hardware amb la resta de l'aplicació és que en un **suposat malfuncionament del hardware, aquest tindria afectació en els dos components**. No només el malfuncionament ha de ser estrictament hardware, sinó que un

comportament no desitjat de la resta de l'aplicació o de la nova eina que afecti al rendiment del hardware de forma negativa, repercuteiria de forma inevitable en l'altre component que comparteix la infraestructura, tenint un únic punt de fallada del sistema.

5.2.2. Integració Desacoblada

Una integració desacoblada suposaria que **el software que implementarà les funcionalitats de la nova eina residiria en una infrastructura externa a la del sistema actual**. Com a premissa, per a poder dur a terme aquest tipus d'integració és obligatori que el codi de la nova eina estigui desacoblat a la del sistema actual. En cas contrari, si el software de la nova eina està acoblat al codi de l'aplicació aquest tipus d'integració hardware no es pot dur a terme, ja que no es pot ubicar el codi de l'eina en una altre infraestructura sense ubicar a la vegada el codi de la resta de l'aplicació. D'aquesta forma, ***una integració desacoblada del hardware implica necessàriament una integració desacoblada del software***.

L'***avantatge*** principal d'aquest tipus d'integració és l'**aïllament total de l'eina de la resta del sistema amb tots els beneficis que això comporta**. D'aquesta forma si aquesta pel motiu que sigui, afecta al rendiment del hardware per un comportament no desitjat, aquest no afectaria al rendiment de la resta de funcionalitats de l'aplicació, ja que els hardwares serien independents i es trobarien aïllats entre si. Això tindria un impacte directe en la mantenibilitat de l'aplicació facilitant-la. Aquest escenari ajudaria a la identificació de l'origen d'errors que afecten al rendiment del sistema d'una forma senzilla. Per altre banda, **no seria necessari aplicar cap canvi en la infraestructura actual quedant la seva configuració actual intacte**. Així es podria dissenyar i configurar el nou hardware a les necessitats de la nova eina sense modificar el hardware on resideix l'aplicació actualment. Aquest fet garantitza l'estabilitat del sistema actual i disminueix l'impacte de la integració, així com distribueix el punt de fallades del sistema fent-lo més robust. Finalment, una altre avantatge es que **el manteniment de la nova infrestructura es pot externalitzar cap a una altre empresa** i, fins i tot, es

podria fer servir sistemes de tipus cloud d'alta disponibilitat per oferir el servei.

El principal ***inconvenient*** d'aquest tipus d'integració és que s'hauria de realitzar una **inversió extra per a poder contractar // comprar hardware nou**. Tot i que l'eina podria a arribar a créixer considerablement en un futur no molt llunyà, en una primera fase del projecte s'oferiria només les funcionalitats bàsiques, pel que podria semblar en principi un dimensionament excessiu del sistema si s'assignen recursos dedicats només a cobrir les necessitats de la nova eina.

5.3. Decisió Tipus Integració

Finalment en aquest apartat s'explica quina decisió s'ha pres referent a com es realitzarà la integració de la nova eina amb la resta del sistema i la seva justificació.

Pel que fa a la **integració software s'ha decidit que l'eina estarà desacoblada al sistema actual**. La raó principal és que des del departament tècnic no es vol seguir implementant funcionalitats acoblades al nucli del sistema, ja que s'ha observat que cada cop que s'implementa una funcionalitat nova, el cost de la mantenibilitat de tota l'aplicació augmenta exponencialment, tal com s'ha explicat en l'apartat d'integració software. Per una altre banda, una integració acoblada amb el software descartaria una possible integració desacoblada del hardware, afectant a la portabilitat del codi nou que es vol implementar i limitant les decisions de disseny del sistema. D'aquesta forma, el codi s'implementarà en un control de versions nou i aquest s'inclourà com a dependència al control de versions de la resta del sistema.

Pel que fa a la integració **hardware, en un principi no es vol invertir en hardware addicional** ja que les funcionalitats que oferirà la nova eina no haurien de tenir un gran impacte en el rendiment, i la infraestructura actual hauria de ser suficient per a cobrir les funcionalitats que oferirà el nou codi. En un futur i tenint el codi desacoblat del software del sistema actual, si es decideix que es vol migrar el codi a una altre infraestructura, l'impacte i el cost de la migració hauria de ser molt baix. D'aquesta forma, el sistema actual i la nova eina conviuran en la mateixa infraestructura hardware.

6. DISSENY DEL SISTEMA

En aquesta secció s'expliquen diferents alternatives relacionades amb el disseny de la nova eina o sistema de gestió de badges.

En primer lloc, s'explica l'**arquitectura del sistema** i es defineix com aquest interaccionarà amb la resta de components així com amb la infraestructura on s'executarà. Després s'explica l'**arquitectura software** que defineix com s'estructuren els components software interns de l'eina. Finalment, es detalla el **disseny del sistema**, identificant els elements necesaris per a implementar l'eina associant-los a components interns software de l'eina detectats en l'apartat d'especificació del sistema.

6.1. Arquitectura del Sistema

En aquest apartat s'explica el procés de disseny de l'arquitectura del sistema la qual està directament relacionada amb el tipus d'integració hardware que es realitzarà. L'arquitectura del sistema definirà el tipus de interfície la qual exposarà les funcionalitats de l'eina per a que components externs puguin utilitzar l'eina.

En l'apartat anterior s'ha pres la decisió de que la nova eina compartirà la infraestructura hardware amb el sistema actual. Però també s'ha comentat que el sistema hauria d'estar preparat per a que si en un futur es vol migrar el codi a un hardware extern, el cost i l'impacte de la migració no sigui elevat. Amb aquesta premissa, **el disseny de l'arquitectura del sistema estarà orientat a reforçar el requisit no funcional de portabilitat**. Un sistema portable és aquell que pot funcionar en diferent tipus d'infraestructura amb un cost i impacte baix en la integració i migració del sistema a un altre hardware.

Per a que la resta de components puguin utilitzar les funcionalitats de la nova eina, aquesta haurà d'oferir les seves funcionalitats com si es tractés d'un servei per a tercers. Es per això que **l'eina caldrà que sigui un servei**, i en concret en aquest apartat es justificarà la necesitat de que sigui **un microservei**. Al llarg d'aquest apartat s'introdueix el concepte de microservei com una evolució de l'arquitectura orientada a serveis. També es

realitza un estudi sobre els tipus d'arquitectures SOA (Service Oriented Architecture) més utilitzats.

6.1.1. Concepte de Microservei

En l'apartat on s'analitzava el tipus d'integració desacoblada del software, s'ha comentat breument que la comunicació entre l'aplicació i la nova eina podria ser com un tipus de contracte. Es a dir, com un preacord entre els dos sistemes per a poder definir els dos elements necessaris per implementar un protocol de comunicació. Aquests són:

- **Signatura dels mètodes**
- **Format de la resposta**

La **signatura dels mètodes** fa referència al punt d'accés de les funcionalitats que ofereix l'eina. Un mètode té un nom com pot ser *createBadge* el qual serveix per crear un badge. I també consta d'uns paràmetres, com poden ser el *nom, la descripció i la imatge del badge* en qüestió. D'aquesta forma el sistema podria invocar aquest mètode informant-li dels paràmetres necessaris per a poder donar d'alta un nou badge.

Pel que fa al **format de la resposta** i seguint amb l'exemple, l'eina podria contestar al sistema amb l'estat de l'operació realitzada, per exemple, que *s'ha pogut crear el badge* o que altrament hi ha hagut algun *error*, i un *identificador* per a que el sistema i l'eina puguin identificar el badge de forma única o pel contrari un *missatge d'error* indicant els motius pel que el procés ha fallat.

Aquest exemple s'hauria d'extrapolar a totes les funcionalitats ofertes per l'eina al sistema. D'aquesta forma es crea un protocol de comunicació entre els dos components.

S'ha de notar dos fets molt importants que permet aquest tipus de comunicació software. Aquests són:

- **Desconeixement de la implementació entre sistemes**
- **Possibilitat d'accés remot**

La primera fa referència a que un sistema no té la necessitat de conèixer els detalls de la implementació del sistema amb el qual es comunica. Poden estar implementats amb tecnologies totalment diferents. Inclús, durant el transcurs de la seva interacció, poden canviar la seva

implementació i, mentre el contracte es respecti, és a dir, la signatura de mètodes no es modifiqui ni el format de la resposta, els sistemes es poden seguir comunicant de la mateixa manera. El contracte de comunicació respon a la pregunta de el què es fa sense necessitat de respondre el com es fa. Es a dir, **un sistema és agnòstic de la implementació del sistema amb el qual es comunica**. En el cas de que es modifiqui el contracte s'ha de notificar per a que la comunicació no es vegi afectada.

La segona fa referència a que **els sistemes** no han de perquè residir en la mateixa infraestructura i **es poden comunicar de forma remota tan sols coneixent les dades de connexió entre ells**. Més concretament, tan sols coneixent la direcció ip o el nom del host remot i el seu port de connexió hauria de ser suficient per a poder establir una connexió remota a través de la xarxa, poder iniciar la comunicació i poder invocar les diferents funcionalitats que ofereix un sistema.

El protocol de comunicació entre sistemes, el desconeixement de la implementació així com la possibilitat de comunicació remota, són els elements bàsics que defineixen un ***software orientat a servei*** [20]. L'objectiu principal és el de poder comunicar sistemes els quals oculten i no comparteixen la seva lògica d'implementació entre ells. És a dir, ***la comunicació entre els sistemes és totalment independent de la tecnologia*** que utilitzen els components per a implementar les seves pròpies funcionalitats. D'aquesta forma es crea una capa d'abstracció reduint l'impacte en el cas de que algun dels sistemes modifiqui algun detall de la implementació per a poder realitzar una funcionalitat concreta.

Microservei [22][23] és un concepte el qual es basa i amplia el concepte de software orientat a serveis. Dóna respostes a les preguntes de **com de gran hauria de ser un servei i com hauria de ser la comunicació entre els sitemes**. Segons la seva definició, els serveis haurien de ser de mida petita i els protocols de comunicació haurien de ser lleugers. El benefici de distribuir diferents responsabilitats del **sistema en diferents i petits serveis incrementa la cohesió i disminueix l'acoblament del sistema**. Aquest fet, fa que modificar i afegir funcionalitats sigui molt més simple i tingui un cost menor. Si el tipus de comunicació utilitza un **protocol lleuger** es **redueix el tràfic de xarxa**. En el context d'una infraestructura on

existeixen diferents microserveis que interactuïn entre ells, l'**ample de banda de la xarxa passa a ser un recurs crític** el qual s'ha de gestionar de la forma més eficient possible. Això s'aconsegueix reduint la quantitat i la mida de la informació intercanviada entre els microserveis.

6.1.2. Avaluació Serveis Web

En aquest apartat es realitza un estudi sobre les diferents tecnologies que permeten implementar software orientat a serveis.

6.1.2.1. XML-RPC

Creat per Dave Winer l'any 1998, XML Remote Procedure Call [23] és un protocol de comunicació entre sistemes el qual permet invocar un mètode implementat de forma remota. El client crea un fitxer de tipus XML del mètode que vol invocar informant dels paràmetres necessaris. El servidor rep la petició, invoca el mètode en qüestió, requerit pel client, i crea un altre fitxer del tipus XML el qual envia al client. La resposta conté en quin estat a finalitzat el procés i pot contenir dades addicionals per la post-gestió del procés per part del client. La comunicació es sol realitzar a través del protocol de xarxa HTTP. És el precursor del protocol SOAP el qual s'explica en el següent apartat.

6.1.2.2. SOAP

Es tracta de l'evolució del protocol XML-RPC. SOAP [24] (Simple Object Access Protocol) és un protocol estàndard per la W3C que defineix com dos objectes en diferents processos poden comunicar-se mitjançant intercanvi de dades XML. A priori la definició és la mateixa que XML-RPC. No obstant aquest protocol afegeix més complexitat que l'anterior. Defineix d'una forma estàndard com ha de ser el format del fitxer XML creat pel client, el qual ha de contenir un sobre (envelope) l'arrel del missatge i el cos (body) el qual conté el contingut del missatge. Es poden definir conjunts de regles de codificació que serveixen per a representar estructures de dades complexes i la convenció per a representar crides a procediments i les seves respostes. Per a poder realitzar el descobriment dels serveis que ofereix un sistema se sol utilitzar el format WSDL (Web Services Description

Language). Es tracta d'un fitxer proporcionat pel servidor on es descriuen els serveis que ofereix, així com el tipus d'estructura de dades que pot processar.

Aquest tipus de protocol és àmpliament utilitzat per implementar software orientat a serveis. S'integra de forma eficient en entorns de programació distribuïda i d'alta concurrència. No obstant i com a inconvenient, el client està molt lligat a la infraestructura del servidor pel que fa al format del protocol, el qual és considerat poc flexible i fràgil als canvis que es puguin realitzar afectant a la comunicació entre sistemes.

6.1.2.3. REST

Al contrari dels 2 protocols explicats en els apartats anteriors, REST [25] (REpresentational State Transfer) és un tipus d'arquitectura, definida en l'any 2000 per Roy Fielding, coautor principal de l'especificació del protocol HTTP. S'utilitza en el context de desenvolupament web totalment recolzada en l'estàndard HTTP. Permet crear serveis i aplicacions que poden ser utilitzades per qualsevol dispositiu o client que es pugi comunicar mitjançant aquest protocol. És més simple i més convencional que les alternatives XML-RPC i SOAP. En aquest tipus d'arquitectura i de forma genèrica, l'accés a serveis es realitza mitjançant URL i els paràmetres s'informen mitjançant mètodes HTTP com poden ser incrustats en la URL (GET) o enviats a través de la xarxa en el cos del missatge (POST). El servidor contesta al client amb el codi d'estat i informació addicional per a que el client pugui realitzar el post-procés. Per potenciar la navegació del client, el servidor sol incloure en la resposta URLs de funcionalitats relacionades. D'aquesta forma, el client tan sols coneixent una única funcionalitat, podria conèixer de forma progressiva la resta de funcionalitats que ofereix el sistema a mesura que interacciona amb aquest. L'analogia seria la d'un usuari que accedeix a un portal web mitjançant una URL. La diferencia és que en aquest context es tracta d'un sistema que navega per les funcionalitats que ofereix l'altre.

6.2. Arquitectura del Software

En aquest apartat es descriu el procés de disseny de l'arquitectura de la nova eina que es desenvoluparà, la qual està directament relacionada amb la integració d'aquesta eina amb els sistemes software d'Atrapalo (veure secció anterior).

En l'apartat on es realitzava l'anàlisi sobre l'impacte de la integració software, s'ha pres la decisió d'integrar l'eina de forma desacoblada al portal web Atrapalo i que es farà com un microservei. El codi de l'eina es crearà en un control de versions independent del de la resta del software del portal web Atrapalo. En aquest context i encara que els dos components comparteixin infraestructura, el portal web Atrapalo accedirà a les funcionalitats de la nova eina de forma remota i independent.

Aquest tipus d'integració ofereix l'oportunitat i la flexibilitat de poder plantejar diferents tipus d'arquitectura per dissenyar el nou software sense estar limitats a l'arquitectura pre-exitent de la resta de l'aplicació.

Al llarg dels següents apartats, es descriurà quines són les característiques desitjables que hauria de complir el disseny de l'arquitectura de la nova eina. Finalment s'analitzen dos tipus d'arquitectures per poder valorar quina és la més adequada.

6.2.1. Objectiu Principal: Codi Desacoblat

S'ha parlat molt en els apartats anteriors del desacoblament i l'acoblament de l'eina amb el portal web Atrapalo, tant a nivell software com hardware, analitzant les seves avantatges i els seus inconvenients.

En aquest apartat es farà encara més èmfasi en aquests conceptes però en un altre context. No és suficient només evitar l'acoblament de l'eina amb la resta del portal web Atrapalo, també **s'ha d'evitar l'acoblament de les diferents parts de l'eina, és a dir entre el que podriem anomenar components interns de la pròpia eina**. El software dissenyat per implementar la nova eina hauria de **complir amb una sèrie de principis per que aquest pugui ser ampliable i mantenible de forma**

senzilla. Si no es segueixen uns certs criteris de correctesa durant el procés de disseny es podria dissenyar un sistema el qual tindria un cost alt de manteniment i d'ampliació. En altres paraules, evitar el desacoblament de l'eina amb la resta del portal web Atrapalo és un primer i important pas, però una manca de criteri durant el disseny de la propia eina podria provocar que es creï una nova eina desacoblada cap a l'exterior però fortament acoblada per dins entre els seus components interns.

Per poder realitzar aquest tasca de la forma més correcta possible i evitar crear un sistema defectuós, es recomana utilitzar els ***principis d'enginyeria del software agrupats en l'acrònim SOLID***, introduïts per Robert C. Martin a principis de la dècada del 2000 [26]. Aquests principis s'enfoquen en realitzar una correcte gestió de les dependències entre components interns d'un sistema. Una correcte gestió significa mantenir els components del sistema desacoblats internament. D'aquesta forma el sistema es manté flexible, robust i reutilizable.

Aquest l'acrònim SOLID agrupa 5 principis els quals es descriuen a continuació:

- **Single responsability principle**

  Un objecte només hauria de tenir una única responsabilitat i realitzar només una tasca.

- **Open / closed principle**

  Les entitats software han de ser obertes per la seva extensió, tancades per la seva modificació.

- **Liskov substitution principle**

  Reemplaçar un objecte per un subtipus d'aquest no ha de modificar el correcte funcionament del programa.

- **Interface segregation principle**

  Moltes interfícies clients són millor que una interfície de propòsit general.

- **Dependency inversion principle**

  Dependre d'abstraccions no d'implementacions.

Encara que a vegades és difícil seguir al peu de la lletra els principis a l'hora de dissenyar un sistema, és molt aconsellable tenir-los presents durant tot el procés disseny. Encara que tots els principis són importants i estan relacionats entre si, en aquest projecte es farà especial èmfasi en el principi de ***Dependency Inversion Principle***, el qual s'explicarà en detall en els següents apartats.

6.2.2. Codi Testejat

Una altre característica fonamental que ha de presentar l'arquitectura del software és que el **codi nou que es crei ha de poder ser testejat de forma automatitzada i unitària** [27]. Aquest fet està directament relacionat amb el desacoblament intern dels components que composen el sistema. Un codi acoblat és molt més difícil de testejar que un codi desacoblat. Per a poder testejar el sistema de forma robusta s'hauria de poder realitzar tests unitaris sobre cada component de forma aïllada de la resta del sistema, tasca improbable de poder dur a terme en un codi acoblat ja que, per definició, no es pot testejar un component de forma aïllada. Per conseqüència, **un codi software que no té tests automatitzats que garantitzin el seu correcte funcionament, és un codi menys mantenible**.

Per un altre banda, els tests també garantitzen que la funcionalitat segueixi comportant-se de forma correcte encara que es modifiqui. Això significa que si existeix la necessitat de modificar el codi, si no es realitza de forma correcte, els tests no funcionaran i informaran al programador que s'ha provocat un error en la modificació del codi, el qual ha de ser corregit. En el cas de que el codi sigui ampliat, s'han de crear nous tests per cobrir el nou software creat.

En conclusió, cada component crític que formi part del nucli del sistema ha d'estar testejat per a garantir el correcte funcionament de la resta de l'aplicació.

6.2.3. Avaluació arquitectures del Software

Una vegada s'ha plantejat de forma genèrica les característiques desitjables que ha de tenir l'arquitectura del software, s'analitzen dos tipus d'arquitectura que poden ser útils per a dissenyar el sistema plantejat.

6.2.3.1. Model Vista Controlador

Es tracta d'un patró d'arquitectura software que procura separar les dades i la lògica de negoci de la interfície d'usuari i el mòdul encarregat de gestionar els esdeveniments i les comunicacions. D'aquesta forma, aquest patró presenta **tres capes** ben diferenciades:

- **Model**
  És la capa encarregada de gestionar les dades.
- **Vista**
  És la capa encarregada d'interaccionar amb l'usuari
- **Controlador**
  És la capa encarregada de processar la informació i de comunicar-se amb la resta de capes.

L'usuari interacciona amb la capa vista la qual envia la informació comunicada a la capa Controlador. El controlador s'encarrega de processar la informació que la capa Vista li ha comunicat i accedeix a la capa Model en el cas que hagi de consultar o persistir alguna dada. Una vegada finalitzat el processament de la informació i l'accés a dades, el Controlador li retorna el resultat a la capa Vista. Finalment, la capa Vista li mostra el resultat a l'usuari.

Una de les restriccions que té aquest tipus d'arquitectura és que **no es poden comunicar directament la capa vista i la capa model sense que intervingui la capa controlador**. D'aquesta forma s'obté un cert desacoblament de l'arquitectura amb capes que tenen rols molt definits, on la Vista compleix amb el rol GUI (Graphical User Interface), el Controlador té la responsabilitat de processar i executar les regles de negoci i la capa Model s'encarrega de la persistència // consulta de dades.

Aquest tipus de patró **potencia el Single Responsability Principle dels principis SOLID**.

6.2.3.2. Arquitectura Hexagonal

Igual que el patró MVC es tracta d'un tipus d'arquitectura que té com a objectiu el desacoblament per capes dels components software que conformen una aplicació. Però va un pas més enllà.

En el cas del MVC, el Controlador podria accedir directament al model de dades, fet que feia necessari que el Controlador coneixes el detall de la implementació i la lògica amb la que està implementada la capa Model. En altres paraules i a mode d'exemple, si la capa Model està implementada amb la tecnologia de base de dades MySQL, el controlador ha d'utilitzar un client de MySQL per comunicar-se amb la capa model, fet que comporta que el controlador hagi de coneixer massa en detall la capa de dades. Addicionalment, no existeix cap mecanisme per a evitar que el controlador pugui processar directament les estructures de dades que li retorna la capa Model, fet que crearia una dependència directa amb el tipus d'estructures de dades que utilitza la capa model per gestionar les seves operacions. En aquest context, ***qualsevol modificació de la capa Model comporta una modificació de la capa Controlador, creant acoblament i en conseqüència augmentant el cost de manteniment de l'aplicació***.

El que es proposa en l'Arquitectura Hexagonal de forma general és que **el Controlador** o la capa encarregada de gestionar la lògica de negoci **rebi per dependència el component el qual serà el responsable de comunicar-se amb la Model, però que el Controlador sigui agnòstic per si mateix i desconegui la implementació de la lògica de la capa Model**. D'aquesta forma, el Controlador en comptes de comunicar-se amb la capa model directament amb el client MySQL, una capa més externa l'injectaria per dependència quin tipus de client ha de fer servir per connectar-se amb la capa Model. D'aquesta forma, es garanteix que la **capa encarregada de la lògica de negoci només processa exclusivament lògica de negoci estan totalment desacoblada de la resta de capes**.

Aquest tipus d'arquitectura ***a part de potenciar el principi SOLID Single Responsability Principle introdueix el principi Dependency Inversion Principle***, és a dir que és millor dependre d'abstraccions que d'implementacions.

6.3. Disseny del Software

En aquest apartat i un cop realitzada l'especificació del sistema, es procedeix a dissenyar el sistema. En aquest apartat es descriu el patró de disseny escollit per implementar els components software. Finalment, es detalla els diagrames de fluxe dels casos d'ús els quals descriuen la interacció dels components per assolir l'objectiu que té cada cas d'ús.

6.3.1. Patró Command Handler

Per a realitzar el disseny i la implementació dels casos d'ús s'utilitza el patró de disseny Command Handler [28]. Per definició el Patró Command Handler, **permet sol·licitar una operació a un objecte sense conèixer realment la seva lògica.** D'aquesta forma, s'encapsula la petició com un tipus d'objecte facilitant la parametrització per a poder executar una funcionalitat concreta.

En el context del projecte, ***els command handlers tindran la lògica de negoci que defineix un cas d'ús en concret***. Per a poder executar aquesta lògica, tots els *command handlers* tindran un únic mètode accessible per part d'altres components el qual tindrà com a nom *handle*. Aquest mètode rebrà com a paràmetre d'entrada un objecte de tipus *command* el qual contindrà tota la informació necessària per a poder executar la lògica del cas d'ús.

Aquest patró reforça el **desacoblament del software aportant abstracció i definició estructurada de components**. Finalment en l'apartat següent s'utiliza la nomenclatura d'aquest patró per a descriure els diagrames de flux dels casos d'ús identificats.

6.3.2. Diagrames de flux

Els diagrames de flux o diagrames de casos d'ús **serveixen per a especificar la comunicació y el comportament d'un sistema**. Serveixen per especificar la comunicació y el comportament d'un sistema mitjançant la seva interacció amb els usuaris o altres sistemes. En concret, mostra la relació entre els actors y els casos d'ús. A continuació es descriuen en detall els diagrames de flux dels casos d'ús identificats així com la interacció entre les classes dels diferents packages.

Casos d'ús de gestió d'usuaris

SignIn

En el cas d'aquest cas d'ús interactuen classes del package Interactor, una classe del package Infrastructure i una altre del package Domain. La classe SignInCommand conté tota la informació que ha informat l'usuari. Aquesta informació es validada per la classe SignInCommandValidator. La classe SignInCommandHandler conté la lógica necesària i utilitza la informació de la classe SignInCommand. Finalment, el SignInCommandHandler crearà una entitat de tipus User i li passarà com a paràmetre d'entrada a la classe UserRepository que engregistrarà l'usuari al sistema.

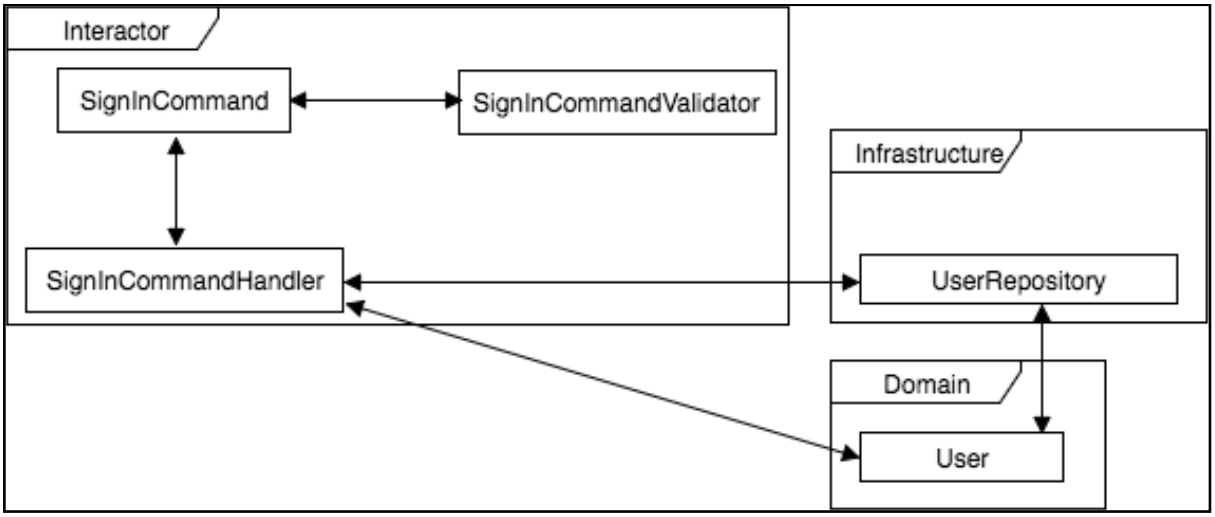


Figura 1 6.3.2 SignIn interacció entre classes de diferents packages

Casos d'ús de gestió d'usuaris

SignIn

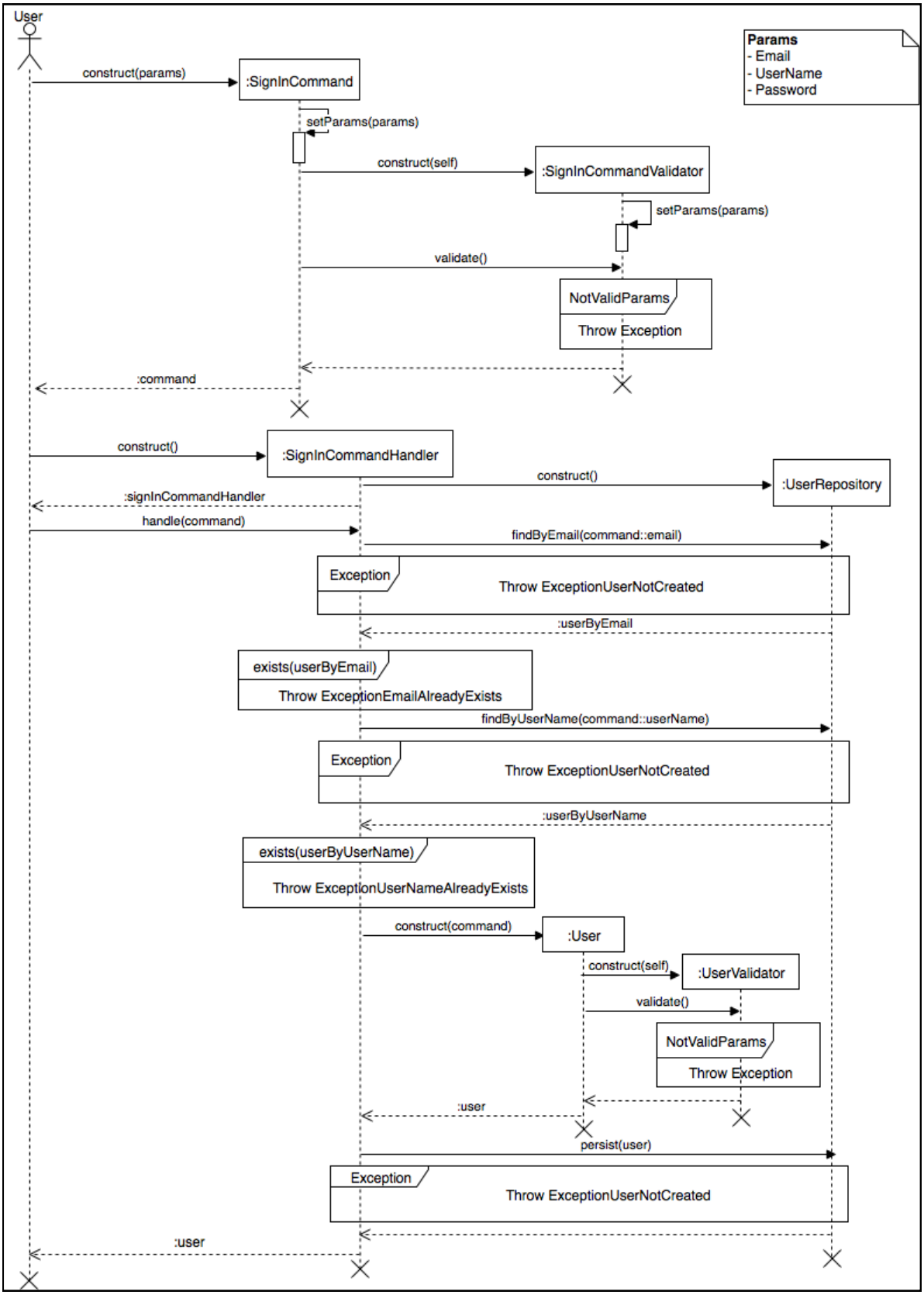


Figura 2 6.3.2 Diagrama de flux cas d'ús SignIn

Escenari Principal

1. L'actor User crea un objecte de tipus SignInCommand passant-li com a paràmetres el Email, el UserName i el PassWord del nou usuari que es desitja fer l'alta.
2. L'objecte SignInCommand s'assigna els paràmetres com a camps propis de la seva classe i crea un objecte de tipus SignInCommandValidator pasant-li el propi SignInCommand com a paràmetre.
3. L'objecte SignInCommand invoca el mètode validate de l'objecte SignInCommandValidator per a que li validi el format i la correctesa dels paràmetres introduïts en el el pas 1.
4. L'actor User crea un objecte de tipus SignInCommandHandler, el qual tindrà la responsabilitat d'executar les regles del cas d'ús.
5. L'objecte SignInCommandHandler crea un objecte de tipus UserRepository el qual conté les dades referents als usuaris donats d'alta al sistema.
6. L'actor User invoca el mètode handle del object SignInCommandHandler passant-li com a paràmetre l'objecte command creat en el pas 1.
7. L'objecte SignInCommandHandler invoca el mètode findByEmail de l'objecte UserRepository passant-li com a paràmetre el Email de l'objecte SignInCommand.
8. L'objecte UserRepository no retorna cap entitat de tipus User indicant que no existeix cap usuari registrat en el sistema amb l'email indicat per l'objecte SigInCommand.
9. L'objecte SignInCommandHandler invoca el métode findByUserName de l'objecte UserRepository passant-li com a paràmetre el UserName de l'objecte SignInCommand.
10. L'objecte UserRepository no retorna cap entitat de tipus User indicant que no existeix cap usuari registrat en el sistema amb l'email indicat per l'objecte SigInCommand.
11. L'objecte SignInCommandHandler crea una entitat de tipus User amb els paràmetres de l'objecte SigInCommand.
12. L'entitat User crea un objecte de tipus UserValidator passant-li com a paràmetre la pròpia entitat User.
13. L'entitat User invoca el métode validate de l'objecte UserValidator per a que li validi el format i la correctesa dels seus camps.
14. L'objecte SignInCommandHandler invoca el mètode persist de l'objecte UserRepository passant-li com a paràmetre l'entitat User creada en el pas 12.
15. L'objecte UserRepository dóna d'alta al sistema l'entitat User creada en el pas 12.
16. L'objecte SignInCommandHandler li retorna l'entitat User creada en el pas 12 l'actor User i acaba el cas d'ús.

Escenaris Excepcionals

3.1 En cas de que algun d'aquests paràmetres no sigui considerat vàlid es llançarà una excepció la qual informarà a l'actor User quin valor no es vàlid, perquè no es vàlid i l'usuari no es crearà.

7.1 En cas de que l'objecte UserRepository llanci una excepció pel motiu que sigui, l'objecte SigInCommandHandler la capturarà i llançarà una excepció informant a l'actor User que l'usuari no s'ha creat.

8.1 En cas de que l'objecte UserRepository retorni una entitat de tipus User, l'objecte SigInCommandHandler llançarà una excepció informant l'actor User que ja existeix un usuari amb el Email de l'objecte SigInCommand i l'usuari no es crearà.

9.1 En cas de que l'objecte UserRepository llanci una excepció pel motiu que sigui, l'objecte SigInCommandHandler la capturarà i llançarà una excepció informant a l'actor User que l'usuari no s'ha creat.

10.1 En cas de que l'objecte UserRepository retorni una entitat de tipus User, l'objecte SigInCommandHandler llançarà una excepció informant l'actor User que ja existeix un usuari amb el UserName de l'objecte SigInCommand i l'usuari no serà creat.

13.1 En cas de que algún d'aquests camps no sigui considerat vàlid es llançarà una excepció la cual informarà a l'actor User quin valor no es vàlid i perque i l'usuari no serà creat.

14.1 En cas de que l'objecte UserRepository llanci una excepció pel motiu que sigui, l'objecte SigInCommandHandler la capturarà i llançarà una excepció informant a l'actor User que l'usuari no s'ha creat.

LogIn

En el cas d'aquest cas d'ús interactuen classes del package Interactor, una classe del package Infrastructure i una classe del package Domain. La classe LogInCommand conté tota la informació que ha informat l'usuari. Aquesta informació es validada per la classe LogInCommandValidator. La classe LogInCommandHandler conté la lógica necesària i utilitza la informació de la classe SignInCommand. Finalment, la classe UserRepository conté tota la informació dels usuaris enregistrats en el sistema necesària per realitzar validacions creant entitats de tipus User.

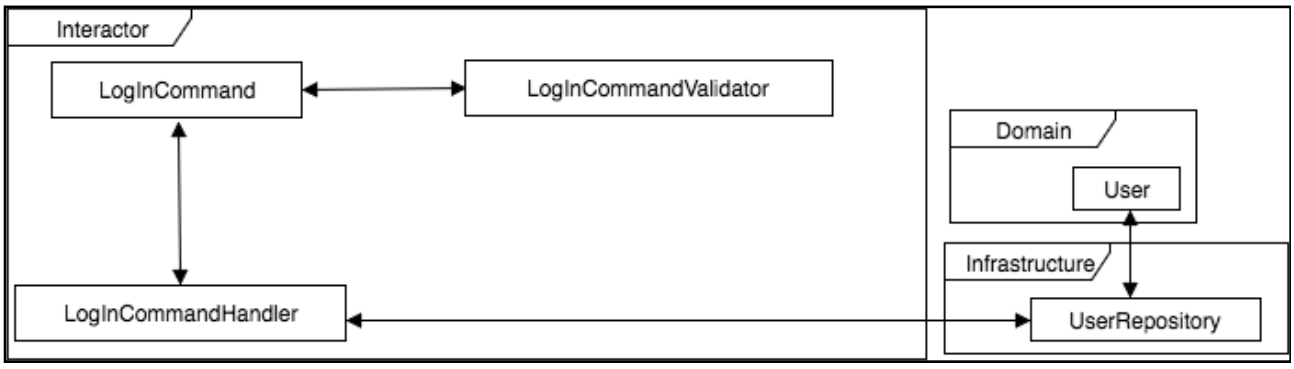


Figura 3 6.3.2 LogIn interacció entre classes de diferents packages

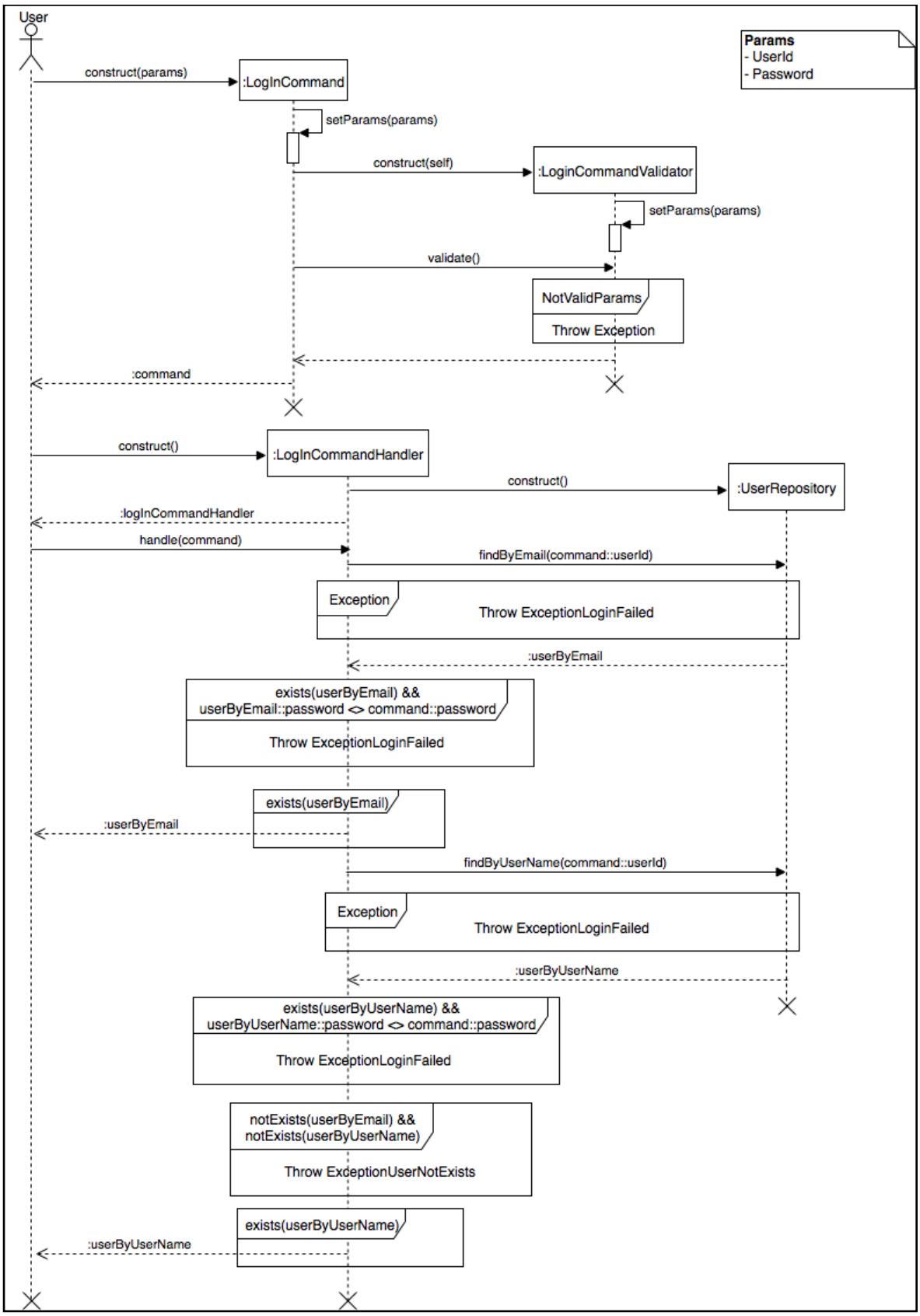


Figura 4 6.3.2 Diagrama de flux cas d'ús LogIn

Escenari Principal

1. L'actor User crea un objecte de tipus LogInCommand passant-li com a paràmetres el UserId i el PassWord del usuari que es desitja donar accés al sistema.
2. L'objecte LogInCommand s'assigna els paràmetres com a camps propis de la seva classe i crea un objecte de tipus LogInCommandValidator passant-li el propi LogInCommand com a paràmetre.
3. L'objecte LogInCommand invoca el mètode validate de l'objecte LogInCommandValidator per a que li validi el format i la correctesa dels paràmetres introduïts en el pas 1.
4. L'actor User crea un objecte de tipus LogInCommandHandler, el qual tindrà la responsabilitat d'executar les regles del cas d'ús.
5. L'objecte LogInCommandHandler crea un objecte de tipus UserRepository el qual conté les dades referents als usuaris donats d'alta al sistema.
6. L'actor User invoca el mètode handle del object LogInCommandHandler pasant-li com a paràmetre l'objecte command creat en el pas 1.
7. L'objecte LogInCommandHandler invoca el mètode findByEmail de l'objecte UserRepository passant-li com a paràmetre el UserId de l'objecte LogInCommand.
8. En cas de que el UserRepository retorni l'entitat de tipus User i el seu PassWord coincideix amb el PassWord de l'objecte LogInCommand, se li dóna accés al usuari al sistema, es retorna l'entitat User a l'actor User i el cas d'ús finalitza.
9. En cas de el UserRepository no retorni cap User, el LogInCommandHandler invoca el mètode findByUserName de l'objecte UserRepository passant-li com a paràmetre el UserId de l'objecte LogInCommand.
10. L'objecte UserRepository no retorna cap entitat de tipus User indicant que no existeix cap usuari registrat en el sistema amb l'email indicat per l'objecte SigInCommand.
11. L'objecte SignInCommandHandler invoca el mètode findByUserName de l'objecte UserRepository passant-li com a paràmetre el UserId de l'objecte SignInCommand.

12. En cas de que l'objecte UserRepository retorni l'entitat de tipus User i el seu PassWord coincideix amb el PassWord de l'objecte LogInCommand, se li dóna accés al usuari al sistema, es retorna l'entitat User a l'actor User i el cas d'ús finalitza.

Escenaris Excepcionals

3.1 En cas de que algun d'aquests paràmetres no sigui considerat vàlid es llançarà una excepció la qual informarà a l'actor User quin valor no es vàlid, perquè no es vàlid i l'accés de l'usuari al sistema no es realitzarà.

7.1 En cas de que l'objecte UserRepository llanci una excepció pel motiu que sigui, l'objecte LogInCommandHandler la capturarà i llançarà una excepció informant a l'actor User que l'accés de l'usuari al sistema ha fallat.

8.1 En cas de que l'objecte UserRepository retorni una entitat de tipus User, però el PassWord del User no coincideix amb el PassWord del LogInCommand, l'objecte LogInCommandHandler llançara una excepció informant a l'actor User que l'accés de l'usuari al sistema ha fallat.

9.1 En cas de que l'objecte UserRepository llanci una excepció pel motiu que sigui, l'objecte LogInCommandHandler la capturarà i llançarà una excepció informant a l'actor User que l'accés de l'usuari al sistema ha fallat.

10.1 En cas de que l'objecte UserRepository retorni una entitat de tipus User, però el PassWord del User no coincideix amb el PassWord del LogInCommand, l'objecte LogInCommandHandler llançarà una excepció informant a l'actor User que l'accés de l'usuari al sistema ha fallat.

10.2 En cas de que l'objecte UserRepository no retorni una entitat de tipus User ni en el pas 7 ni en el pas 11, l'objecte LogInCommandHandler llançarà una excepció informant a l'actor User que l'accés de l'usuari no existeix en el sistema.

Casos d'ús de gestió badges

CreateBadge

En el cas d'aquest cas d'ús interactuen classes del package Interactor, classes del package Infrastructure i classes del package Domain. La classe CreateBadgeCommand conté tota la informació que ha informat l'usuari. Aquesta informació es validada per la classe CreateBadgeCommandValidator, la classe UserDataValidator i la classe ImageDataValidator. La classe CreateBadgeCommandHandler conté la lógica necesària i utilitza la informació de la classe CreateBadgeCommand per a crear entitats de tipus Image i Badge. Finalment, les classes UserRepository, ImageRepository i BadgeRepository serveixen per consultar les dades enregistrades dels usuaris i per enregistrar en el sistema les entitats de tipus Image i de tipus Badge.

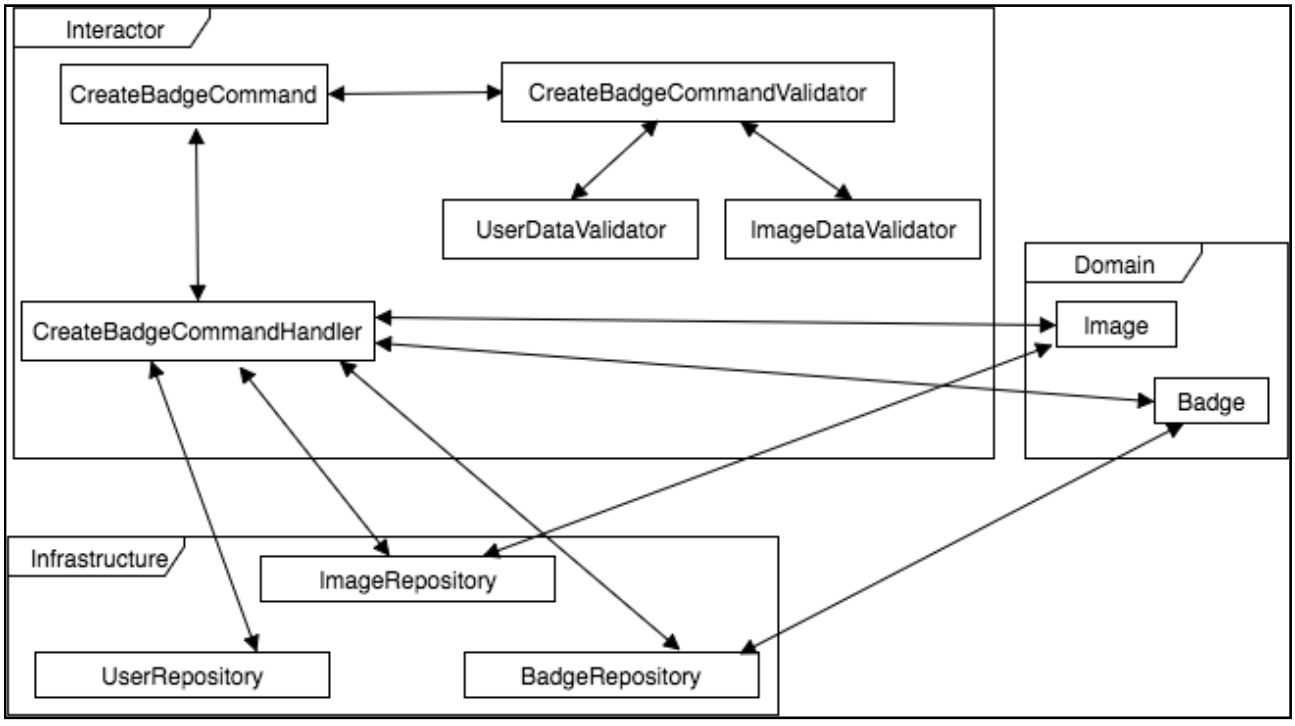


Figura 5 6.3.2 CreateBadge interacció entre classes de diferents packages

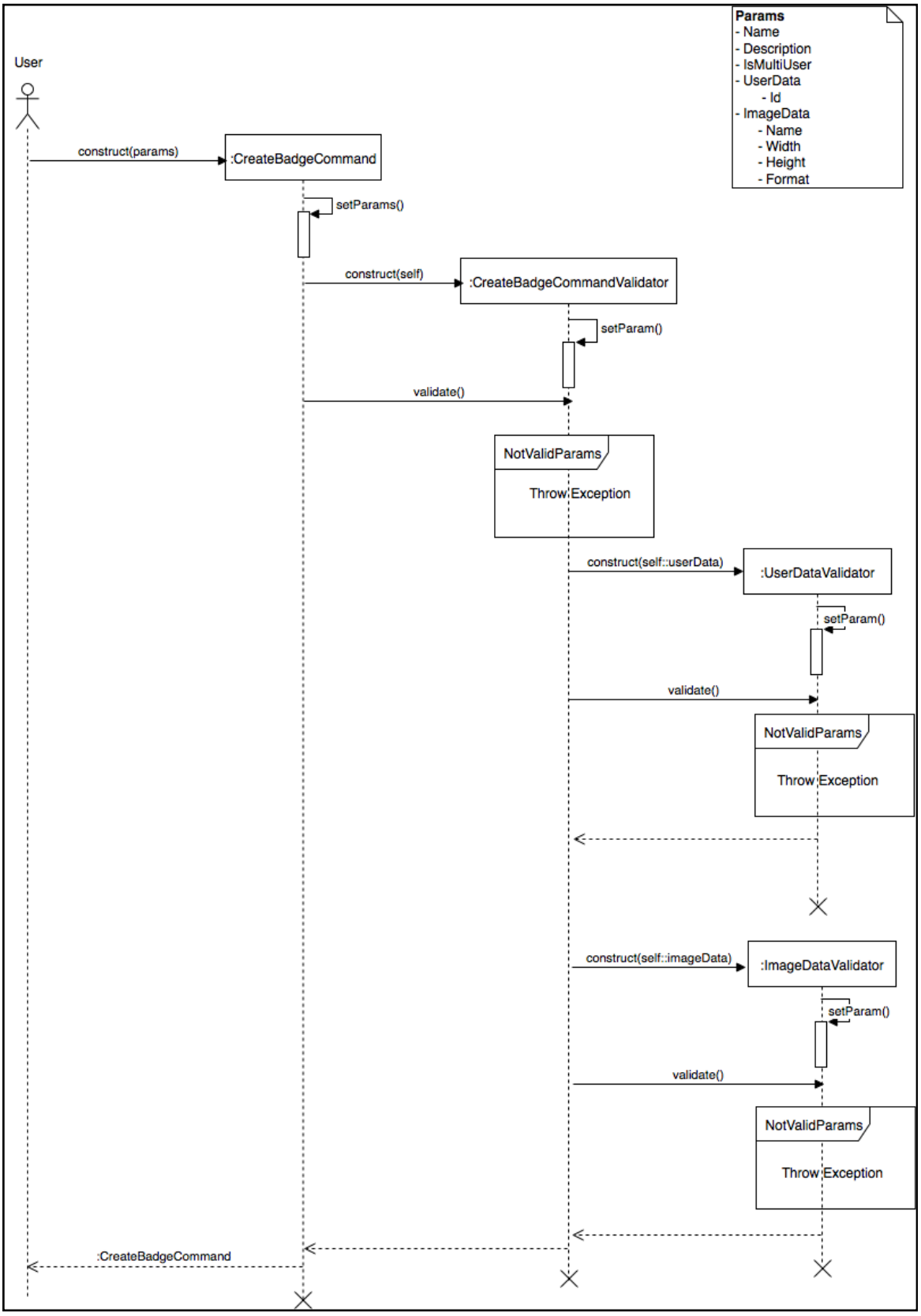


Figura 6 6.3.2 Diagrama de flux cas d'ús CreateBadge

Escenari Principal

1. L'actor User crea un objecte de tipus CreateBadgeCommand passant-li com a paràmetres la següent informació:
   Name, Description, IsMultiUser
   UserData: Id
   ImageData: Name, Width, Height y Format
2. L'objecte CreateBadgeCommand s'assigna els paràmetres com a camps propis de la seva classe i crea un objecte de tipus CreateBadgeCommandValidator passant-li el propi CreateBadgeCommand com a paràmetre.
3. L'objecte CreateBadgeCommand invoca el mètode validate de l'objecte CreateBadgeCommandValidator per a que li validi el format i la correctesa dels paràmetres Name, Description, IsMultiUser introduïts en el pas 1.
4. L'objecte CreateBadgeCommandValidator crea un objecte de tipus UserDataValidator passant-li com a paràmetre l'objecte UserData per a que li validi el seu format i la seva correctesa.
5. L'objecte CreateBadgeCommandValidator invoca el mètode validate de l'objecte UserDataValidator per a que li validi el format i la correctesa del Id de l'objecte UserData introduït en el pas 1.
6. L'objecte CreateBadgeCommandValidator crea un objecte de tipus ImageDataValidator passant-li com a paràmetre l'objecte ImageData per a que li validi el seu format i la seva correctesa.
7. L'objecte CreateBadgeCommandValidator invoca el mètode validate de l'objecte ImageDataValidator per a que li validi el format i la correctesa del Name, Width, Heigth y Format de l'objecte ImageData introduït en el pas 1.

Escenaris Excepcionals

3.1 En cas de que algun d'aquests paràmetres no sigui considerat vàlid es llançarà una excepció la qual informarà a l'actor User quin valor no es vàlid, perquè no es vàlid i no es crearà el Badge.

5.1 En cas de que algun d'aquests paràmetres no sigui considerat vàlid es llançarà una excepció la qual informarà a l'actor User quin valor no es vàlid, perquè no es vàlid i no es crearà el Badge.

7.1 En cas de que algun d'aquests paràmetres no sigui considerat vàlid es llançarà una excepció la qual informarà a l'actor User quin valor no es vàlid, perquè no es vàlid i no es crearà el Badge.

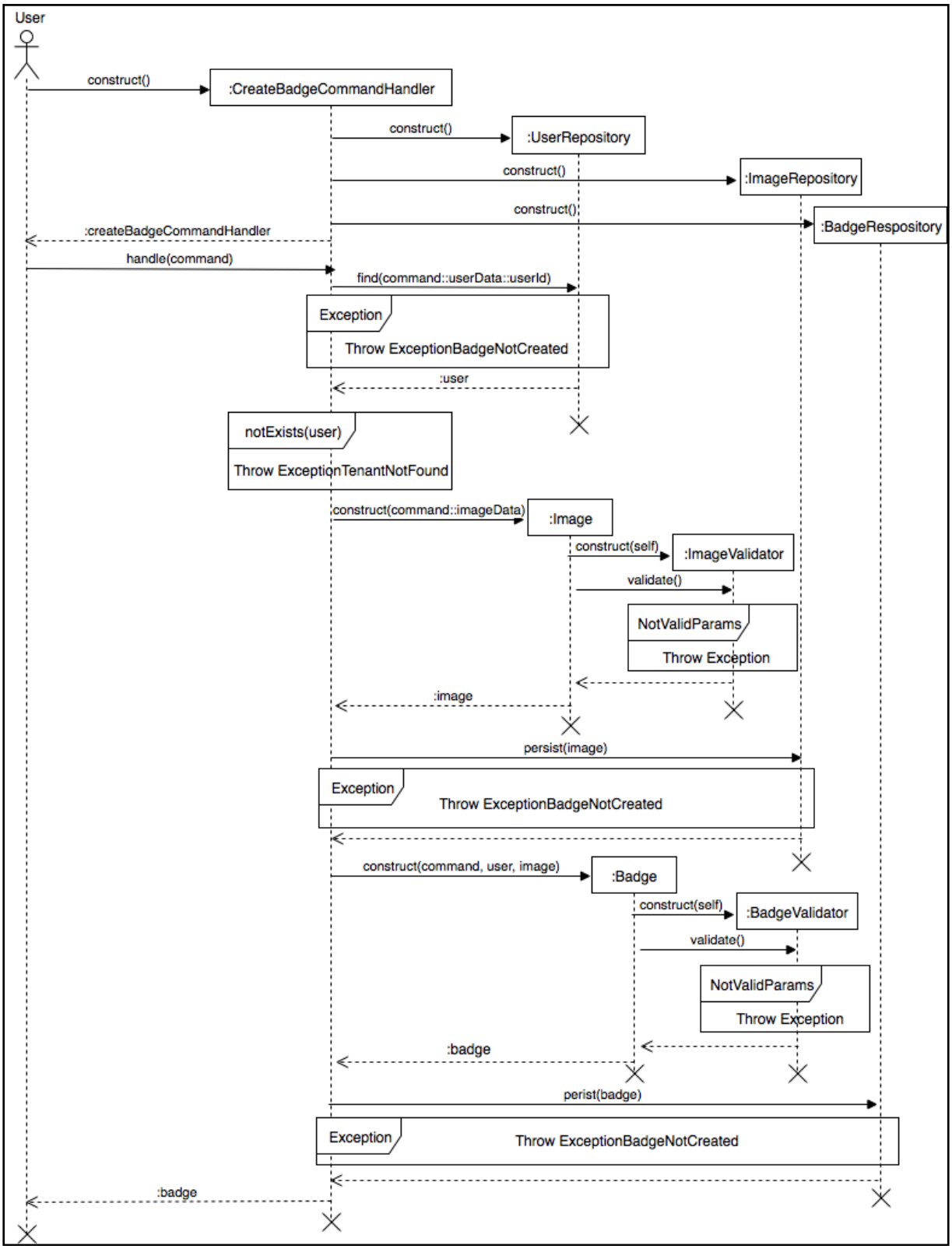


Figura 7 6.3.2 Diagrama de flux cas d'ús CreateBadge

Escenari Principal

1. L'actor User crea un objecte de tipus CreateBadgeCommandHandler, el qual tindrà la responsabilitat d'executar les regles del cas d'ús.
2. L'objecte CreateBadgeCommandHandler crea 3 objectes de tipus UserRepository, ImageRepository i BadgeRepository.
3. L'actor User invoca el mètode handle del object CreateBadgeCommandHandler passant-li com a paràmetre l'objecte command creat en el pas 8.
4. L'objecte CreateBadgeCommandHandler invoca el mètode find de l'objecte UserRepository pasant-li com a paràmetre l'Id de l'objecte UserData inclós en l'objecte CreateBadgeCommand.
5. L'objecte UserRepository retorna una entitat de tipus User.
6. L'objecte CreateBadgeCommandHandler crea una entitat de tipus Image amb els paràmetres de l'objecte ImageData inclòs en l'objecte CreateBadgeCommand.
7. L'entitat Image crea un objecte de tipus ImageValidator passant-li com a paràmetre la pròpia entitat Image.
8. L'entitat Image invoca el mètode validate de l'objecte ImageValidator per a que li validi el format i la correctesa dels seus camps.
9. L'objecte CreateBadgeCommandHandler invoca el mètode persist de l'objecte ImageRepository passant-li com a paràmetre l'entitat Image creada en el pas 15.
10. L'objecte ImageRepository dóna d'alta al sistema l'entitat Image creada en el pas 15.
11. L'objecte CreateBadgeCommandHandler crea una entitat de tipus Badge amb els paràmetres Name, Description, IsMultiUser de l'objecte CreateBadgeCommand, l'entitat User obtinguda en el pas 12 i l'entitat Image creada en el pas 15.
12. L'entitat Badge crea un objecte de tipus BadgeValidator passant-li com a paràmetre la pròpia entitat Badge.
13. L'entitat Badge invoca el mètode validate de l'objecte BadgeValidator per a que li validi el format i la correctesa dels seus camps.
14. L'objecte CreateBadgeCommandHandler invoca el mètode persist de l'objecte BadgeRepository passant-li com a paràmetre l'entitat Badge creada en el pas 18.
15. L'objecte BadgeRepository dóna d'alta al sistema l'entitat Badge creada en el pas 18.
16. L'objecte CreateBadgeCommandHandler li retorna l'entitat Badge creada en el pas 18 l'actor User i acaba el cas d'ús.

Escenaris Excepcionals

11.1 En cas de que l'objecte UserRepository llanci una excepció pel motiu que sigui, l'objecte CreateBadgeCommandHandler la capturarà i llançarà una excepció informant a l'actor User que el badge no s'ha creat.

12.1 En cas de que l'objecte UserRepository no retorni una entitat de tipus User, l'objecte CreateBadgeCommandHandler llançarà una excepció informant a l'actor User que no existeix el User amb l'Id de l'objecte UserData inclós en l'objecte CreateBadgeCommand i el Badge serà creat.

13.1 En cas de que algun d'aquests camps no sigui considerat vàlid es llançarà una excepció la qual informarà a l'actor User quin valor no es vàlid, perquè no es vàlid i el Badge no serà creat.

14.1 En cas de que l'objecte ImageRepository llanci una excepció pel motiu que sigui, l'objecte CreateBadgeCommandHandler la capturará i llançará una excepció informant a l'actor User que el Badge no s'ha creat.

20.1 En cas de que algun d'aquests camps no sigui considerat vàlid es llançarà una excepció la qual informarà a l'actor User quin valor no es vàlid, perquè no es vàlid i el Badge no serà creat.

21.1 En cas de que l'objecte BadgeRepository llanci una excepció pel motiu que sigui, l'objecte CreateBadgeCommandHandler la capturarà i llançarà una excepció informant a l'actor User que el Badge no s'ha creat.

GetBadge

En el cas d'aquest cas d'ús interactuen classes del package Interactor, una classe del package Infrastructure i una classe del package Domain. La classe GetBadgeCommand conté tota la informació que ha informat l'usuari. Aquesta informació es validada per la classe GetBadgeCommandValidator. La classe GetBadgeCommandHandler conté la lógica necesària i utilitza la informació de la classe GetBadgeCommand. Finalment, la classe BadgeRepository conté tota la informació dels badges enregistrats en el sistema necesària per a obtenir l'entitat de tipus Badge demanada per l'usuari.

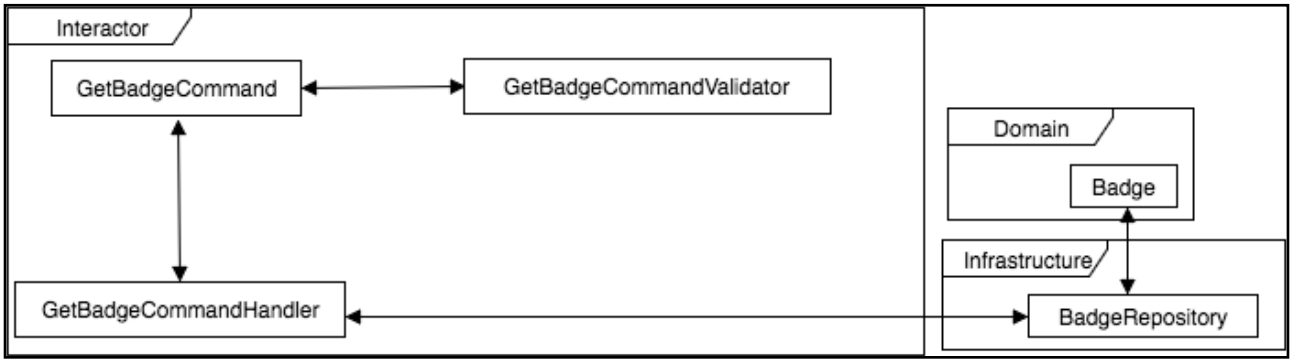


Figura 8 6.3.2 GetBadge interacció entre classes de diferents packages

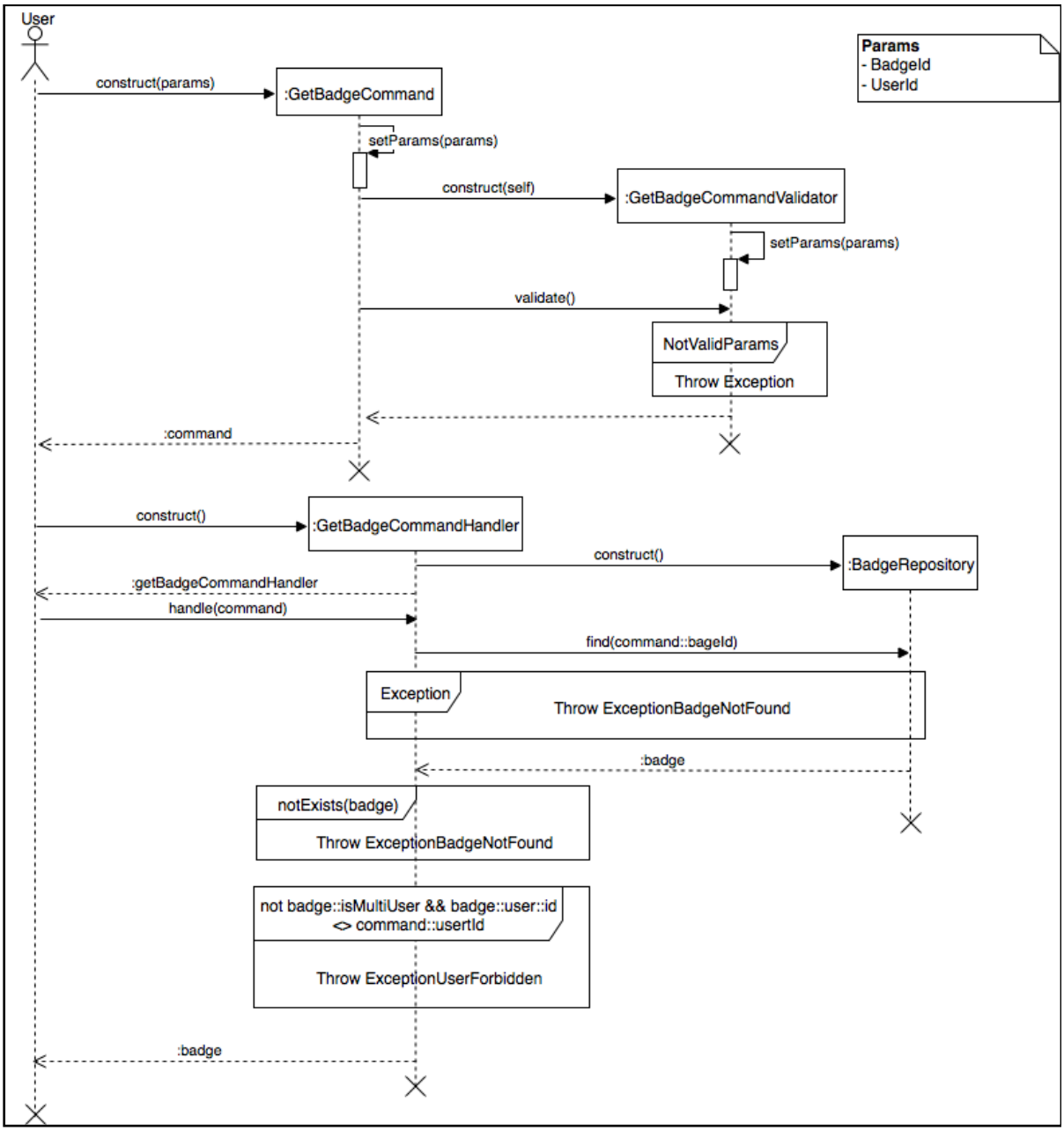


Figura 9 6.3.2 Diagrama de flux cas d'ús GetBadge

Escenari Principal

1. L'actor User crea un objecte de tipus GetBadgeCommand passant-li com a paràmetres el BadgeId i el UserId del Badge que es desitja recuperar.
2. L'objecte GetBadgeCommand s'assigna els paràmetres com a camps propis de la seva classe i crea un objecte de tipus GetBadgeCommandValidator passant-li el propi GetBadgeCommand com a paràmetre.
3. L'objecte GetBadgeCommand invoca el mètode validate de l'objecte GetBadgeCommandValidator per a que li validi el format i la correctesa dels paràmetres introduïts en el pas 1.
4. L'actor User crea un objecte de tipus GetBadgeCommandHandler, el qual tindrà la responsabilitat d'executar les regles del cas d'ús.
5. L'objecte GetBadgeCommandHandler crea un objecte de tipus BadgeRepository el qual conté les dades referents als Badges donats d'alta al sistema.
6. L'actor User invoca el mètode handle del object GetBadgeCommandHandler passant-li com a paràmetre l'objecte command creat en el pas 1.
7. L'objecte GetBadgeCommandHandler invoca el mètode find de l'objecte BadgeRepository passant-li com a paràmetre el BadgeId de l'objecte GetBadgeCommand.
8. L'objecte BadgeRepository retorna la entitat Badge a l'objecte GetBadgeCommandHandler.
9. Si existeix l'entitat Badge i coincideixen el camp Id de l'entitat User relacionada amb l'entitat Badge o l'entitat Badge es multiuser, se li retornat la entitat Badge a l'actor User i el cas d'ús finalitza.

Escenaris Excepcionals

3.1 En cas de que algun d'aquests paràmetres no sigui considerat vàlid es llançarà una excepció la qual informarà a l'actor User quin valor no es vàlid, perquè no es vàlid i no es retornarà cap entitat de tipus Badge.

7.1 En cas de que l'objecte BadgeRepository llanci una excepció pel motiu que sigui, l'objecte GetBadgeCommandHandler la capturarà i llançarà una excepció informant a l'actor User que no s'ha trobat cap Badge amb el valor del paràmetre BadgeId.

9.1 En el cas de que l'objecte BadgeRepository no retorni una entitat de tipus Badge, l'objecte GetBadgeCommandHandler llançarà una excepció informant a l'actor User que no s'ha trobat cap Badge amb el BadgeId informat en l'objecte GetBadgeCommand.

9.2 En el cas de que no coincideixi el UserId de l'objecte GetBadgeCommand amb el Id de l'entitat User relacionada amb l'entitat Badge i que l'entitat Badge no sigui multiuser, l'objecte GetBadgeCommandHandler llançarà una excepció informant a l'actor User que no te permisos per consultar l'entitat Badge.

UpdateBadge

En el cas d'aquest cas d'ús interactuen classes del package Interactor, classes del package Infrastructure i classes del package Domain. La classe UpdateBadgeCommand conté tota la informació que ha informat l'usuari. Aquesta informació es validada per la classe UpdateBadgeCommandValidator, la classe UserDataValidator i la classe ImageDataValidator. La classe UpdateBadgeCommandHandler conté la lógica necesària i utilitza la informació de la classe UpdateBadgeCommand per actualitzar entitats de tipus Image i Badge. Finalment, les classes UserRepository, ImageRepository i BadgeRepository serveixen per consultar les dades enregistrades dels usuaris i per actualitzar en el sistema les entitats de tipus Image i de tipus Badge.

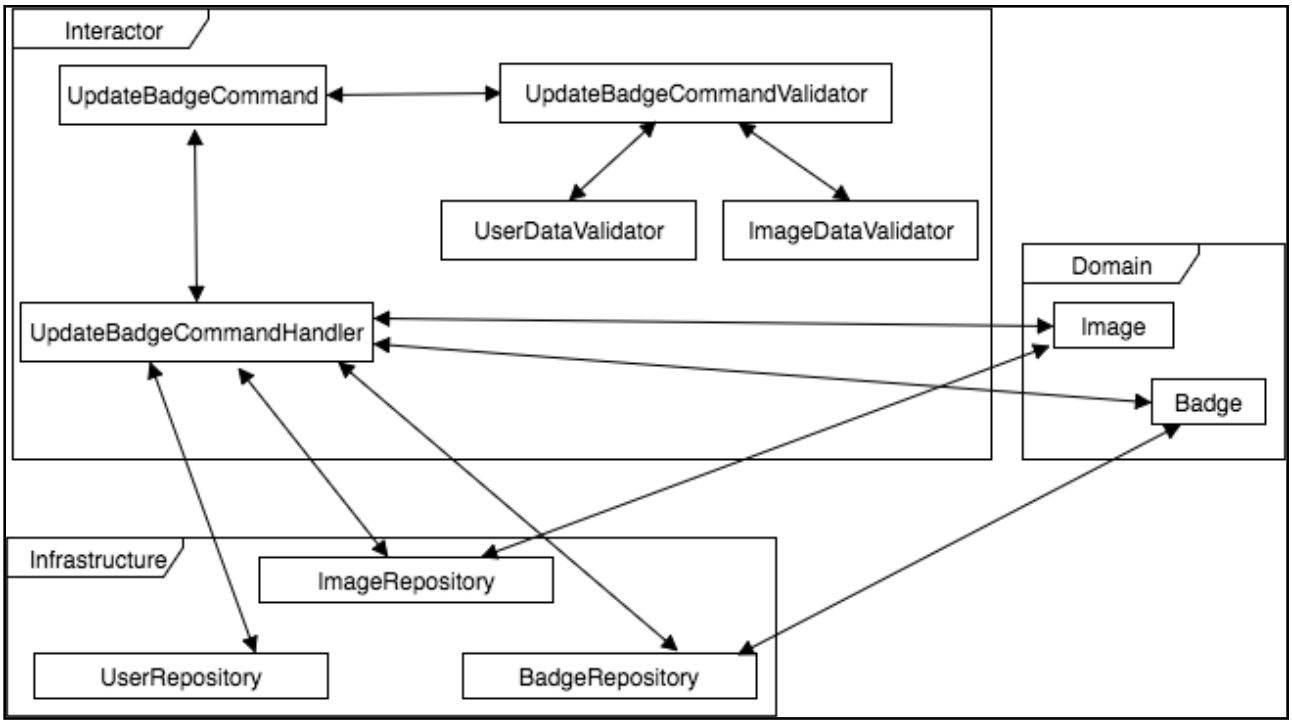


Figura 10 6.3.2 UpdateBadge interacció entre classes de diferents packages

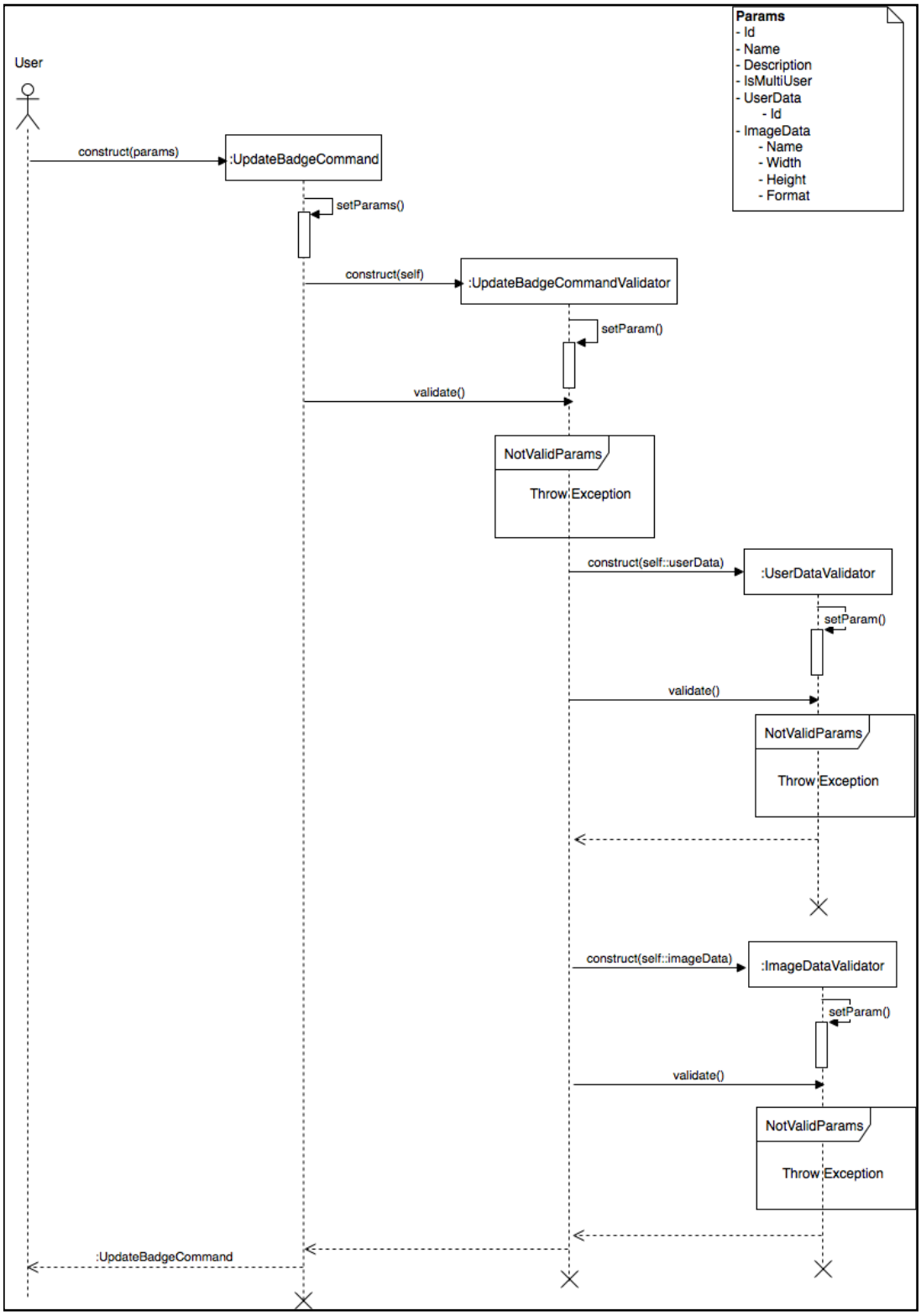


Figura 11 6.3.2 Diagrama de flux cas d'ús UpdateBadge

Escenari Principal

1. L'actor User crea un objecte de tipus UpdateBadgeCommand passant-li com a paràmetres la següent informació:
   Id, Name, Description, IsMultiUser
   UserData: Id
   ImageData: Name, Width, Height y Format
2. L'objecte UpdateBadgeCommand s'assigna els paràmetres com a camps propis de la seva classe i crea un objecte de tipus UpdateBadgeCommandValidator passant-li el propi UpdateBadgeCommand com a paràmetre.
3. L'objecte UpdateBadgeCommand invoca el mètode validate de l'objecte UpdateBadgeCommandValidator per a que li validi el format i la correctesa dels paràmetres Id, Name, Description, IsMultiUser introduïts en el pas 1.
4. L'objecte UpdateBadgeCommandValidator crea un objecte de tipus UserDataValidator passant-li com a paràmetre l'objecte UserData per a que li validi el seu format i la seva correctesa.
5. L'objecte UpdateBadgeCommandValidator invoca el mètode validate de l'objecte UserDataValidator per a que li validi el format i la correctesa del Id de l'objecte UserData introduït en el pas 1.
6. L'objecte UpdateBadgeCommandValidator crea un objecte de tipus ImageDataValidator passant-li com a paràmetre l'objecte ImageData per a que li validi el seu format i la seva correctesa.
7. L'objecte UpdateBadgeCommandValidator invoca el mètode validate de l'objecte ImageDataValidator per a que li validi el format i la correctesa del Name, Width, Heigth y Format de l'objecte ImageData introduït en el pas 1.

Escenaris Excepcionals

3.1 En cas de que algun d'aquests paràmetres no sigui considerat vàlid es llançarà una excepció la qual informarà a l'actor User quin valor no es vàlid, perquè no es vàlid i no es modificarà el Badge.

5.1 En cas de que algun d'aquests paràmetres no sigui considerat vàlid es llançarà una excepció la qual informarà a l'actor User quin valor no es vàlid, perquè no es vàlid i no es modificarà el Badge.

7.1 En cas de que algun d'aquests paràmetres no sigui considerat vàlid es llançarà una excepció la qual informarà a l'actor User quin valor no es vàlid, perquè no es vàlid i no es modificarà el Badge.

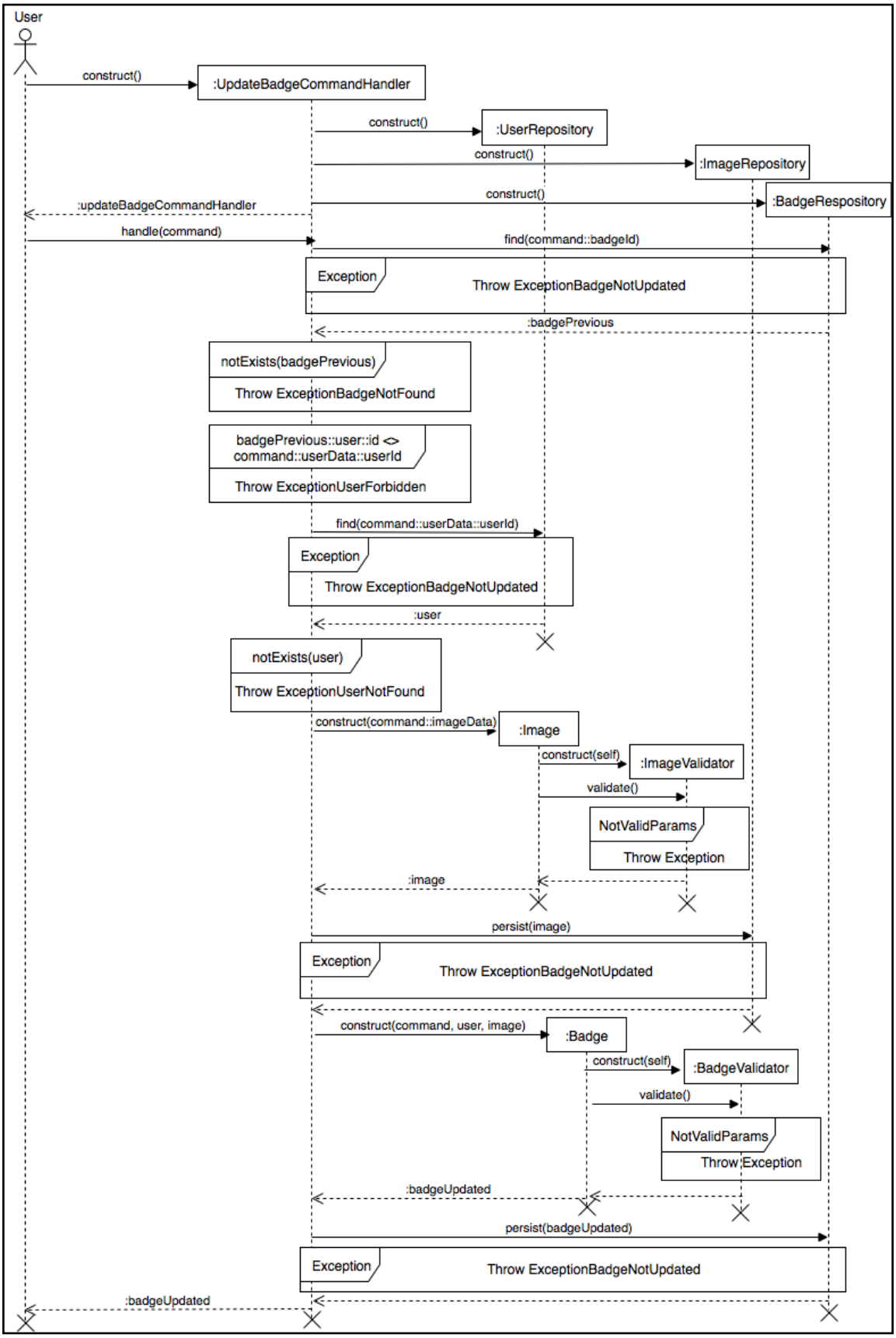


Figura 12 6.3.2 Diagrama de flux cas d'ús UpdateBadge

Escenari Principal

8. L'actor User crea un objecte de tipus UpdateBadgeCommandHandler, el cual tindrà la responsabilitat d'executar les regles del cas d'ús.
9. L'objecte UpdateBadgeCommandHandler crea 3 objectes de tipus UserRepository, ImageRepository i BadgeRepository.
10. L'actor User invoca el mètode handle del object UpdateBadgeCommandHandler pasant-li com a paràmetre l'objecte command creat en el pas 1.
11. L'objecte UpdateBadgeCommandHandler invoca el mètode find de l'objecte BadgeRepository pasant-li com a paràmetre l'Id de l'objecte CreateBadgeCommand.
12. L'objecte BadgeRepository retorna una entitat de tipus Badge.
13. L'objecte UpdateBadgeCommandHandler invoca el mètode find de l'objecte UserRepository pasant-li com a paràmetre l'Id de l'objecte UserData inclós en l'objecte UpdateBadgeCommand.
14. L'objecte UserRepository retorna una entitat de tipus User.
15. L'objecte UpdateBadgeCommandHandler crea una entitat de tipus Image amb els paràmetres de l'objecte ImageData inclós en l'objecte UpdateBadgeCommand i l'Id de l'entitat Image relacionada amb l'entitat Badge.
16. L'entitat Image crea un objecte de tipus ImageValidator passant-li com a paràmetre la propia entitat Image.
17. L'entitat Image invoca el mètode validate de l'objecte ImageValidator per a que li validi el format i la correctesa dels seus camps.
18. L'objecte UpdateBadgeCommandHandler invoca el mètode persist de l'objecte ImageRepository pasant-li com a paràmetre l'entitat Image creada en el pas 15.
19. L'objecte ImageRepository modifica l'entitat Image donada d'alta al sistema i creada en el pas 15.
20. L'objecte UpdateBadgeCommandHandler crea una entitat de tipus Badge amb els paràmetres Id, Name, Description, IsMultiUserde l'objecte CreateBadgeCommand, l'entitat User obtinguda en el pas 14 i l'entitat Image creada en el pas 15.
21. L'entitat Badge crea un objecte de tipus BadgeValidator passant-li com a paràmetre la propia entitat Badge.
22. L'entitat Badge invoca el mètode validate de l'objecte BadgeValidator per a que li validi el format i la correctesa dels seus camps.
23. L'objecte UpdateBadgeCommandHandler invoca el mètode persist de l'objecte BadgeRepository pasant-li com a paràmetre l'entitat Badge creada en el pas 20.
24. L'objecte BadgeRepository modifica l'entitat Badge donada d'alta al sistema i creada en el pas 20.
25. L'objecte UpdateBadgeCommandHandler li retorna l'entitat Badge creada en el pas 20 l'actor User i acaba el cas d'ús.

Escenaris Excepcionals

11.1 En cas de que l'objecte BadgeRepository llanci una excepció pel motiu que sigui, l'objecte UpdateBadgeCommandHandler la capturará i llançará una excepció informant a l'actor User que el badge no s'ha modificat.

12. En cas de que l'objecte BadgeRepository no retorni una entitat de tipus Badge, l'objecte UpdateBadgeCommandHandler llançarà una excepció informant a l'actor User que no existeix el Badge amb l'Id de l'objecte UpdateBadgeCommand i el Badge no serà modificat.

12.2 En cas que no coincideixen l'Id de la entitat User relacionat amb la entitat Badge amb el Id de l'objecte UserData inclós en l'objecte UpdateCommand, l'objecte UpdateCommandHandler llançarà una excepció informant a l'actor User que no te suficients permisos per modificar el Badge no serà modificat.

13.1 En cas de que l'objecte UserRepository llanci una excepció pel motiu que sigui, l'objecte UpdateBadgeCommandHandler la capturará i llançará una excepció informant a l'actor User que el badge no s'ha modificat.

14.1 En cas de que l'objecte UserRepository no retorni una entitat de tipus User, l'objecte UpdateBadgeCommandHandler llançarà una excepció informant a l'actor User que no existeix el User amb l'Id de l'objecte UserData inclós en l'objecte UpdateBadgeCommand i el Badge no serà modificat.

17.1 En cas de que algún d'aquests camps no sigui considerat vàlid es llançarà una excepció la cual informarà a l'actor User quin valor no es vàlid, perque no es vàlid i el Badge no serà modificat.

18.1 En cas de que l'objecte ImageRepository llanci una excepció pel motiu que sigui, l'objecte UpdateBadgeCommandHandler la capturará i llançará una excepció informant a l'actor User que el Badge no s'ha modificat.

22.1 En cas de que algún d'aquests camps no sigui considerat vàlid es llançarà una excepció la cual informarà a l'actor User quin valor no es vàlid, perque no es vàlid i el Badge no serà modificat.

23.1 En cas de que l'objecte BadgeRepository llanci una excepció pel motiu que sigui, l'objecte UpdateBadgeCommandHandler la capturará i llançará una excepció informant a l'actor User que el Badge no s'ha modificat.

ListBadges

En el cas d'aquest cas d'ús interactuen classes del package Interactor, dos classes del package Infrastructure i dos classes del package Domain. La classe ListBadgesCommand conté tota la informació que ha informat l'usuari. Aquesta informació es validada per la classe ListBadgesCommandValidator. La classe ListBadgesCommandHandler conté la lógica necesària i utilitza la informació de la classe ListBadgesCommand. Finalment, la classe BadgeRepository conté tota la informació dels badges enregistrats en el sistema necesària per a obtenir el llistat de Badges demanada per l'usuari i la classe UserRepository conté la informació dels usuaris enregistrats en el sistema per a poder realitzar validacions d'accés.

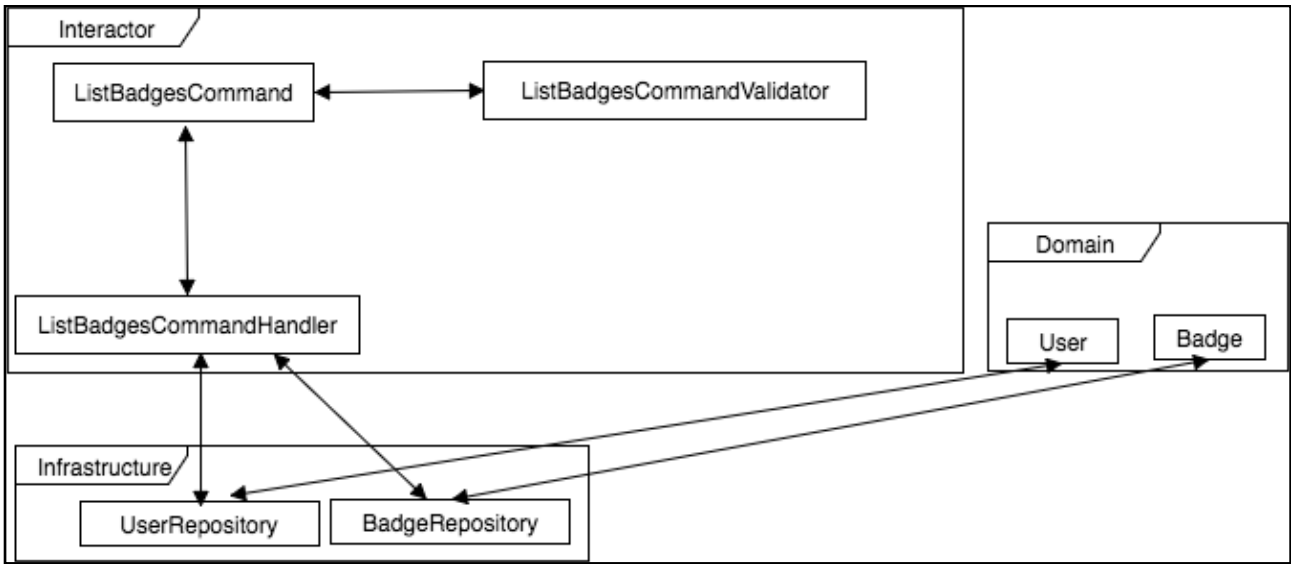


Figura 13 6.3.2 ListBadges interacció entre classes de diferents packages

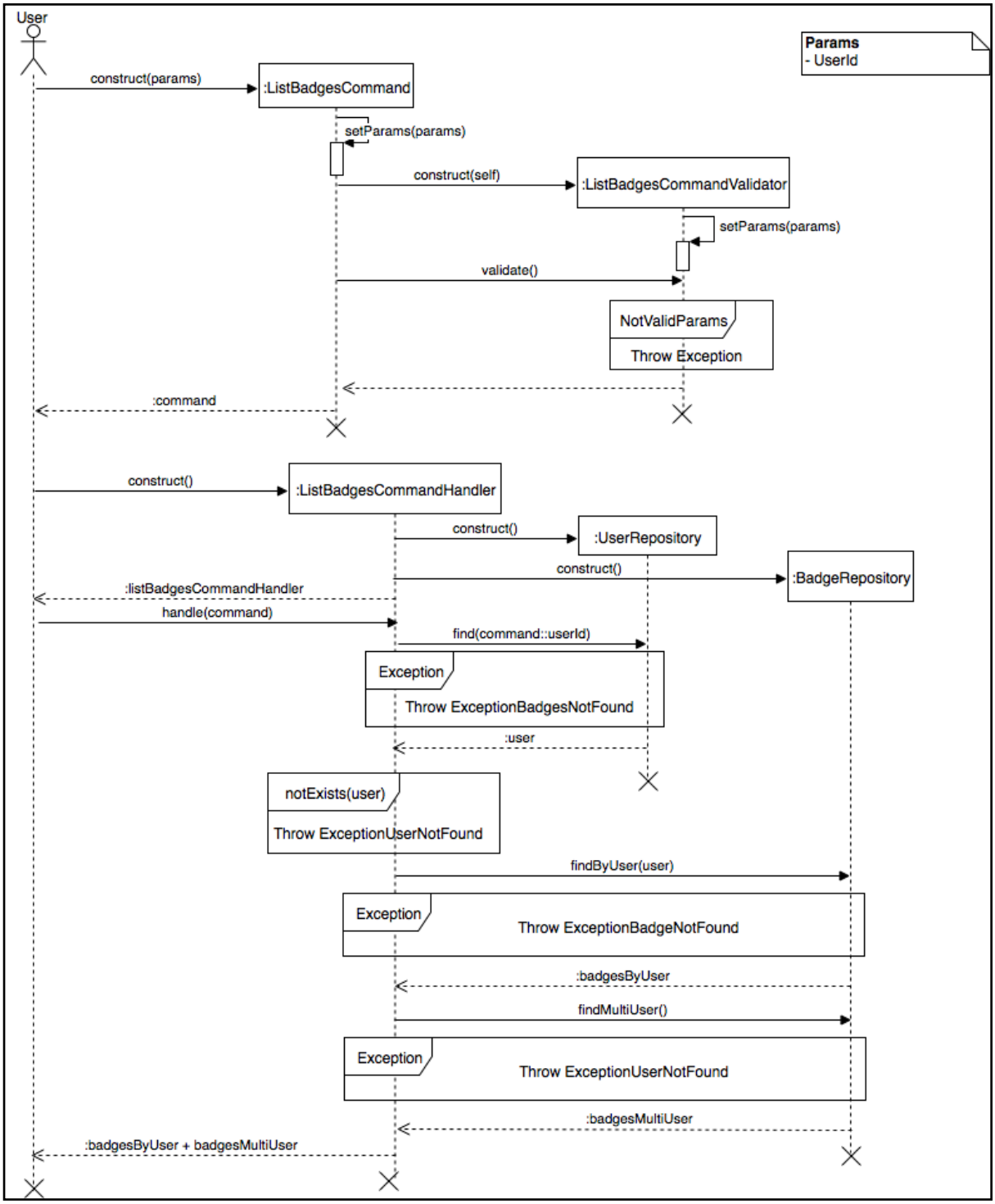


Figura 14 6.3.2 Diagrama de flux cas d'ús ListBadges

Escenari Principal

1. L'actor User crea un objecte de tipus ListBadgesCommand passant-li com a paràmetres el UserId per recuperar els seus Badges i els que siguin multiuser.
2. L'objecte ListBadgesCommand s'assigna els paràmetres com a camps propis de la seva classe i crea un objecte de tipus ListBadgesCommandValidator passant-li el propi ListBadgesCommand com a paràmetre.
3. L'objecte ListBadgesCommand invoca el mètode validate de l'objecte ListBadgesCommandValidator per a que li validi el format i la correctesa dels paràmetres introduïts en el pas 1.
4. L'actor User crea un objecte de tipus ListBadgesCommandHandler, el qual tindrà la responsabilitat d'executar les regles del cas d'ús.
5. L'objecte ListBadgesCommandHandler crea 2 objectes de tipus UserRepository y BadgeRepository.
6. L'actor User invoca el mètode handle del object ListBadgesCommandHandler passant-li com a paràmetre l'objecte command creat en el pas 1.
7. L'objecte ListBadgesCommandHandler invoca el mètode find de l'objecte UserRepository passant-li com a paràmetre l'Id de l'objecte ListBadgesCommand.
8. L'objecte UserRepository retorna una entitat de tipus User.
9. L'objecte ListBadgesCommandHandler invoca el mètode findByUser de l'objecte BadgeRepository passant-li com a paràmetre l'entitat User obtinguda en el pas 8. Retornarà les entitats Badge de les quals la entitat User es propietari.
10. L'objecte BadgeRepository retorna una llista de entitats Badge o una llista buida a l'objecte GetBadgeCommandHandler.
11. L'objecte ListBadgesCommandHandler invoca el mètode findMultiUser de l'objecte BadgeRepository. Retornarà les entitats Badge les quals son compartides per a totes les entitats User del sistema.
12. L'objecte BadgeRepository retorna una llista de entitats Badge o una llista buida a l'objecte ListtBadgesCommandHandler.
13. L'objecte ListBadgesCommandHandler retorna a l'actor User un llistat de Badges format pels elements dels llistats trobats en els passos 10 i 12 i el cas i el cas d'ús finalitza.

Escenaris Excepcionals

3.1 En cas de que algun d'aquests paràmetres no sigui considerat vàlid es llançarà una excepció la qual informarà a l'actor User quin valor no es vàlid, perquè no es vàlid i no es retornarà cap entitat de tipus Badge.

7.1 En cas de que l'objecte UserRepository llanci una excepció pel motiu que sigui, l'objecte ListBadgesCommandHandler la capturarà i llançarà una excepció informant a l'actor User que no s'ha trobat cap entitat de tipus Badge.

8.1 En cas de que l'objecte UserRepository no retorni una entitat de tipus User, l'objecte ListBadgesCommandHandler llançarà una excepció informant a l'actor User que no existeix el User amb l'Id de l'objecte ListBadgesCommand i no es retornarà cap entitat de tipus Badge.

9.1 En cas de que l'objecte BadgeRepository llanci una excepció pel motiu que sigui, l'objecte ListBadgesCommandHandler la capturarà i llançarà una excepció informant a l'actor User que no s'ha trobat que no s'ha trobat cap entitat de tipus Badge.

11.1 En cas de que l'objecte BadgeRepository llanci una excepció pel motiu que sigui, l'objecte ListBadgesCommandHandler la capturarà i llançarà una excepció informant a l'actor User que no s'ha trobat que no s'ha trobat cap entitat de tipus Badge.

DeleteBadge

En el cas d'aquest cas d'ús interactuen classes del package Interactor, dos classes del package Infrastructure i dos classes del package Domain. La classe DeleteBadgeCommand conté tota la informació que ha informat l'usuari. Aquesta informació es validada per la classe DeleteBadgeCommandValidator. La classe DeleteBadgeCommandHandler conté la lógica necesària i utilitza la informació de la classe DeleteBadgeCommand. Finalment, la classe BadgeRepository conté tota la informació dels badges enregistrats en el sistema necesària per a esborrar el Badge demanat per l'usuari i la classe ImageRepository conté la informació de les imatges enregistrades en el sistema per a poder esborrar la imatge associada al badge.

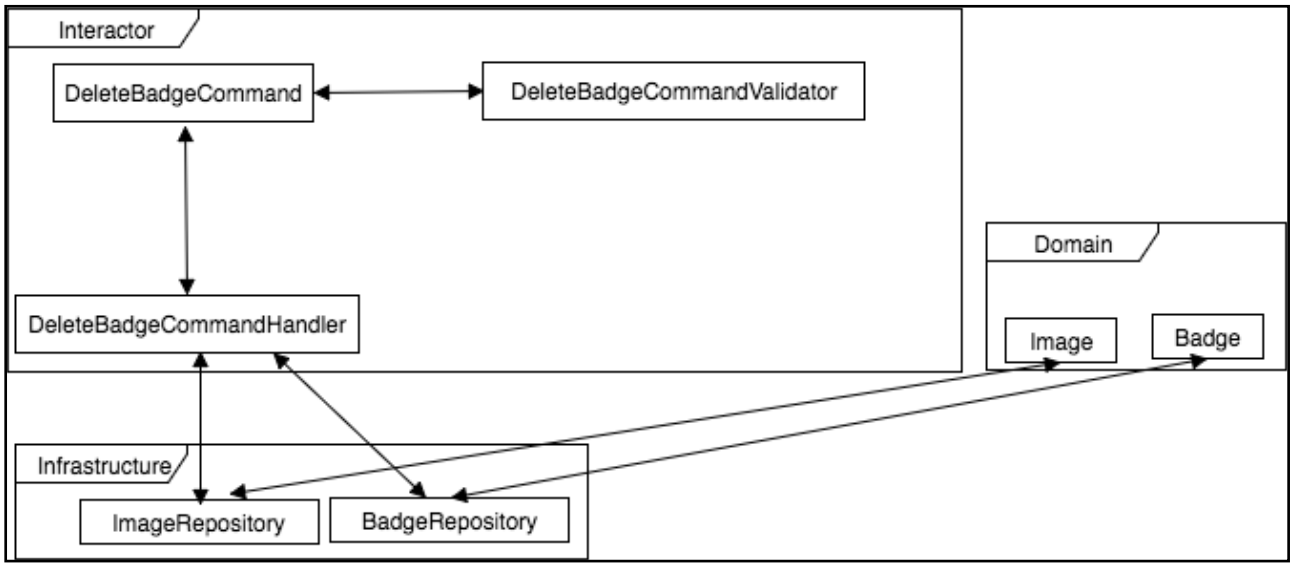


Figura 15 6.3.2 DeleteBadge interacció entre classes de diferents packages

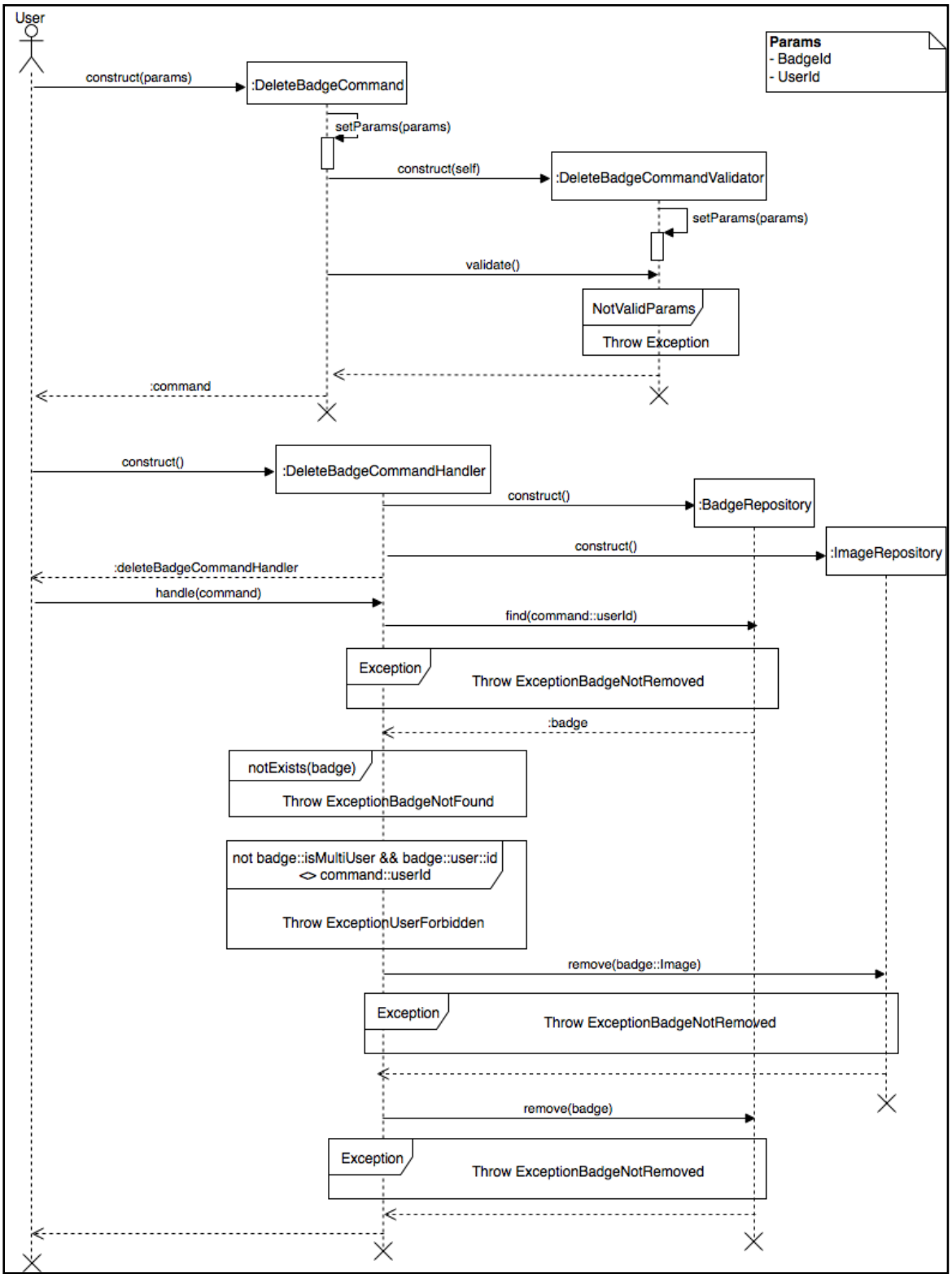


Figura 16 6.3.2 Diagrama de flux cas d'ús DeleteBadge

Escenari Principal

1. L'actor User crea un objecte de tipus DeleteBadgeCommand passant-li com a paràmetres el BadgeId i el UserId del Badge que es desitja eliminar.
2. L'objecte DeleteBadgeCommand s'assigna els paràmetres com a camps propis de la seva classe i crea un objecte de tipus DeleteBadgeCommandValidator passant-li el propi DeleteBadgeCommand com a paràmetre.
3. L'objecte DeleteBadgeCommand invoca el mètode validate de l'objecte DeleteBadgeCommandValidator per a que li validi el format i la correctesa dels paràmetres introduïts en el pas 1.
4. L'actor User crea un objecte de tipus DeleteBadgeCommandHandler, el cual tindrà la responsabilitat d'executar les regles del cas d'ús.
5. L'objecte DeleteBadgeCommandHandler crea 2 objectes de tipus BadgeRepository y ImageRepository.
6. L'actor User invoca el mètode handle del object DeleteBadgeCommandHandler passant-li com a paràmetre l'objecte command creat en el pas 1.
7. L'objecte DeleteBadgeCommandHandler invoca el mètode find de l'objecte BadgeRepository pasant-li com a paràmetre el BadgeId de l'objecte DeleteBadgeCommand.
8. L'objecte BadgeRepository retorna l'entitat Badge a l'objecte DeleteBadgeCommandHandler.
9. L'objecte DeleteBadgeCommandHandler invoca el mètode remove de l'objecte ImageRepository passant-li com a paràmetre l'entitat Image relacionada amb l'entitat Badge obtinguda en el pas 8.
10. L'objecte ImageRepository elimina l'entitat Image del sistema passada com a paràmetre en el pas 9.
11. L'objecte DeleteBadgeCommandHandler invoca el mètode remove de l'objecte BadgeRepository passant-li com a paràmetre l'entitat Badge relacionada amb l'entitat Badge obtinguda en el pas 8.
12. L'objecte BadgeRepository elimina l'entitat Badge del sistema passada com a paràmetre en el pas 11.
13. Es retorna el fluxe al l'actor User i el cas d'ús finalitza.

Escenaris Excepcionals

3.1 En cas de que algun d'aquests paràmetres no sigui considerat vàlid es llançarà una excepció la qual informarà a l'actor User quin valor no es vàlid, perquè no es vàlid i no es retornarà cap entitat de tipus Badge.

7.1 En cas de que l'objecte BadgeRepository llanci una excepció pel motiu que sigui, l'objecte DeleteBadgeCommandHandler la capturarà i llançarà una excepció informant a l'actor User que no s'ha eliminat cap Badge amb el valor del paràmetre BadgeId informat en l'objecte DeleteBadgeCommand.

8.1 En el cas de que l'objecte BadgeRepository no retorni una entitat de tipus Badge, l'objecte DeleteBadgeCommandHandler llançarà una excepció informant a l'actor User que no s'ha trobat cap Badge amb el BadgeId informat en l'objecte DeleteBadgeCommand.

8.2 En el cas de que no coincideixi el UserId de l'objecte DeleteBadgeCommand amb el Id de l'entitat User relacionada amb l'entitat Badge i que l'entitat Badge no sigui multiuser, l'objecte DeleteBadgeCommandHandler llançarà una excepció informant a l'actor User que no te permisos per eliminar l'entitat Badge.

9.1 En cas de que l'objecte ImageRepository llanci una excepció pel motiu que sigui, l'objecte DeleteBadgeCommandHandler la capturarà i llançarà una excepció informant a l'actor User que no s'ha eliminat cap Badge amb el valor del paràmetre BadgeId informat en l'objecte DeleteBadgeCommand.

11.1 En cas de que l'objecte BadgeRepository llanci una excepció pel motiu que sigui, l'objecte DeleteBadgeCommandHandler la capturarà i llançarà una excepció informant a l'actor User que no s'ha eliminat cap Badge amb el valor del paràmetre BadgeId informat en l'objecte DeleteBadgeCommand.

6.4. Decisions de Disseny

En aquest apartat és comenten les decisions que s'han pres durant el procés de disseny. Aquestes decisions constituiran la base de la implementació del sistema.

L'***arquitectura del sistema serà API Rest***. Per a poder crear un microservei és la millor opció de les analitzades. Aquest tipus d'arquitectura permet l'accés remot per part del client a les funcionalitats que ofereix el sistema. El protocol utilitzat per a la comunicació és lleuger i robust ja que es tracta del protocol HTTP el qual es utilitzat àmpliament per molts sistemes en la xarxa. Addicionalment, permet adaptar el format de la comunicació entre client i sistema en cada cas en particular, sense la necessitat d'utilitzar un format de comunicació propi de l'arquitectura com és en el cas de de SOAP.

Pel que fa a l'***arquitectura del software s'utilitzarà l'arquitectura hexagonal***. És el tipus d'arquitectura ideal per potenciar el desacoblament dels components interns els quals formen part del sistema, un dels objectius principals del procés de disseny. Cal recordar que aquesta arquitectura potencia el Single Responsability Principle (un component té una única responsabilitat) i el Dependency Inversion Principle (dependència d'abstraccions, no de implementacions), els quals garantitzen la correctesa del disseny desacoblat.

Finalment, per implementar els ***casos d'ús*** identificats o la lògica de negoci, s'utilitzarà el ***patró de disseny Command Handler***. Aquest patró ofereix la possibilitat d'encapsular en un objecte de tipus Command tota la informació necessària per a que un Command Handler pugui executar la lògica de negoci d'un cas d'ús en concret. Aquest patró, a part d'oferir més desacoblament entre components, ofereix organització i estructura per a poder realitzar una implementació robusta i ordenada del sistema.

7. IMPLEMENTACIÓ DEL SISTEMA

En aquest apartat s'aplicaran les decisions de disseny preses fins ara per poder dur a terme la implementació del sistema. **L'objectiu principal serà el d'implementar un sistema desacoblat per a reduir el cost de manteniment i reforçar la portabilitat del sistema**. Els conceptes claus per a poder dur a terme aquesta tasca seran:

- Arquitectura hexagonal
- Injecció de dependències
- Tests unitaris
- API Rest

7.1. Arquitectura Hexagonal

Alistair Cockburn va inventar el concepte l'any 2005 [29]. Es tracta d'una resposta al desig de voler crear aplicacions que siguin testejables. La seva definició és la següent:

"Aquesta arquitectura permet que una aplicació pugui ser utilitzada per igual tant per usuaris, com per programes, tests automatitzats o scripts, i serà implementada i testejada aïllada dels diferents dispositius on es pugui executar i de les bases de dades finals"

D'aquesta forma, l'arquitectura hexagonal o Ports-And-Adapters [30] té com a ***objectiu desacoblar el nucli de l'aplicació de la tecnologia emprada***. Aquest fet permet:

- Serveis de diferents tipus o tecnologia puguin ser connectats al nucli o core de l'aplicació
- El nucli de l'aplicació pot funcionar sense la dependència directe de cap servei ni de cap tecnologia

El ***nucli lògic o la lògica de negoci*** d'una aplicació consisteix en els **algoritmes que són essencials** per aconseguir un propòsit. Aquests implementen els **casos d'ús**. Si per qualsevol fet o situació es canvien, s'està canviant l'essència de l'aplicació.

Els ***serveis***, per contra, ***no són essencials***. **Poden ser reemplaçats sense que canviï el propòsit o essència de l'aplicació**. Exemples de serveis poden ser l'accés a la base de dades i d'altre tipus d'emmagatzematge, interfícies d'usuari (GUI), e-mail i altres components de comunicació o dispositius hardwares. Fins i tot, el framework és considera un d'aquests serveis.

Els avantatges d'aquesta arquitectura són:

- El nucli de l'aplicació pot ser testejat independentment dels serveis existents
- Es facilita el fet de poder canviar uns serveis per uns altres segons la situació i les necessitats ho requereixin

7.1.1. Ports And Adapters

A continuació es pot veure un esquema d'aquest tipus d'arquitectura:

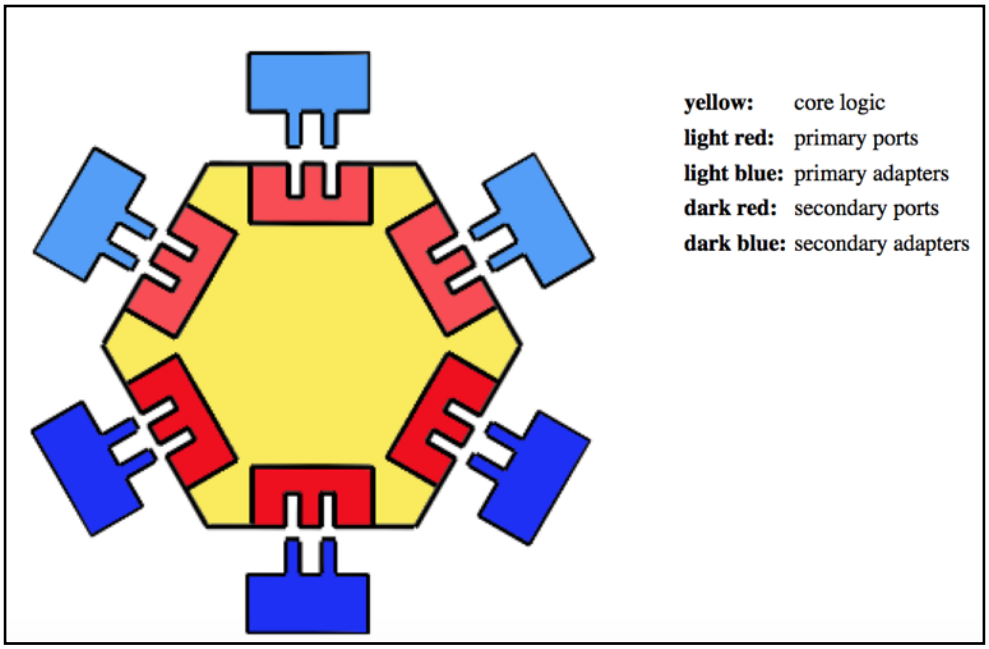


Figura 1 6.1.1 Ports and Adapters

Un ***Port*** representa un **punt d'entrada al nucli del sistema**. Defineix un conjunt de funcions ofertes pel nucli del sistema als components externs.

A continuació es detallen els diferents tipus de ports:

- **Ports Primaris**

  Es tracten dels principals punts d'entrada al nucli de l'aplicació. Aquests són utilitzats pels adapters primaris que es troben en els components que interactuen amb l'usuari, externs a la lògica de negoci. Permeten modificar objectes, atributs i relacions. La seva invocació permet l'execució de la lògica de negoci, on es troben implementats els casos d'ús.

- **Ports Secundaris**

  Són les interfícies pels adapters secundaris. Aquests són utilitzats pel nucli de l'aplicació. Un exemple de port secundari és una interfície que persisteix un únic objecte. Aquesta interfície simplement especifica que un objecte serà creat, recuperat, modificat o eliminat. **No exposa en cap cas com s'ha de realitzar aquest tipus d'accions**. És com un tipus de **contracte** de les accions que es poden realitzar amb aquest tipus de servei. Una especificació sense implementació.

Un ***Adapter*** és un **accés entre l'aplicació i el servei requerit per l'aplicació**. Està relacionat directament amb un port específic.

A continuació es detallen els diferents tipus d'adapters:

- **Adapter Primari**

  Representa una **peça de codi la qual es troba ubicada en l'arquitectura entre l'usuari i el nucli lògic**. Un adapter d'aquest tipus podria ser un test unitari pel nucli de l'aplicació o un controlador el qual interactua tant amb el nucli del sistema com amb l'usuari, sent un pont entre el nucli del sistema i l'usuari. **L'adapter primari invoca les funcionalitats que ofereix el nucli del sistema, és a dir, els ports primaris**.

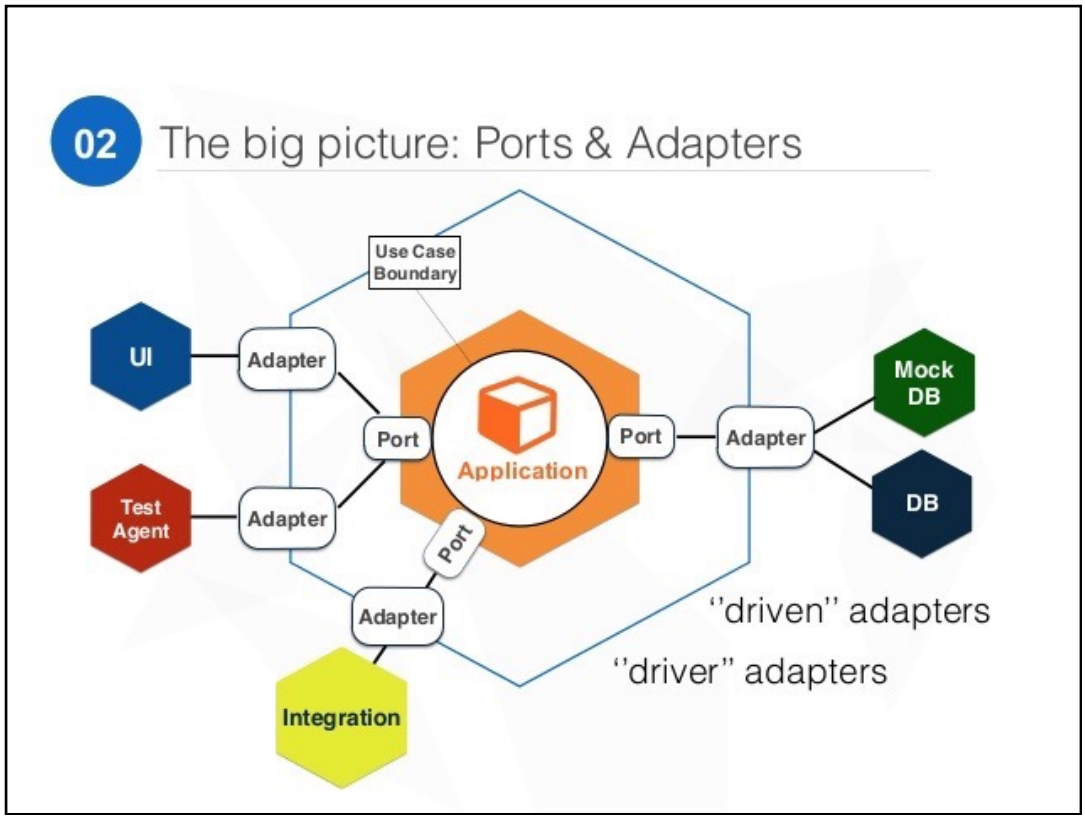


Figura 2 6.1.1 Exemple Ports and Adapters

- **Adapter Secundari**

Es tracta d'una i**mplementació del port secundari**. Es pot tractar d'una aplicació que recupera informació d'una base de dades amb els **paràmetres d'entrada i format de sortida que especifica el port secundari**. També es pot tractar d'un objecte fake per a poder testejar certes parts del codi del nucli del sistema, sense necessitat d'utilitzar cap tipus de tecnologia. **El nucli del sistema o lògica de negoci invoca les funcions de l'adapter secundari**.

Aquest tipus d'arquitectura és molt recomanable per assolir l'objectiu d'obtenir el 100% de cobertura de codi per part dels tests i per dotar a l'aplicació de la facilitat de poder canviar el medi d'emmagatzematge o altre tipus de codi de tercers de forma freqüent, reduint dràsticament el cost de mantenibilitat. Finalment, l'aplicació pot ser utilitzada per diferents tipus d'usuaris i sistemes, on cada un d'aquests pot crear la seva pròpia variant de l'aplicació, tan sols enllaçant les seves pròpies implementacions dels adapters adequada a les seves necessitats.

A continuació es descriu el flux general d'una aplicació construïda basant-se amb aquest tipus d'arquitectura:

- Una instància de l'aplicació és creada, així com els seus adapters primaris.
- Els adapters secundaris són passats al nucli del sistema com a dependències. És el que s'anomena **injecció de dependències**.
- Els adapters primaris reben la connexió al nucli del sistema i comencen a fer funcionar l'aplicació.
- L'entrada de dades que realitza l'usuari, és processada per un o varis adapters primaris i li passa al nucli lògic.
- Durant el processament de les dades, el nucli del sistema invoca mètodes dels adaptadors secundaris prèviament injectats.
- La resposta del nucli lògic és retornada cap als adaptadors primaris els quals li retornen, a la vegada, a l'usuari.

Aquest tipus d'arquitectura permet aconseguir dissenyar sistemes amb les següents característiques:

- **Independents dels frameworks**

  L'arquitectura no depèn de l'existència de certa llibreria. Aquest fet permet utilitzar els frameworks com a eines evitant que les seves limitacions influeixin en el disseny del sistema.
- **Sistemes testejables**

  La lògica de negoci pot ser testejada sense interfície d'usuari (UI), base de dades, servidor web o qualsevol altre element extern depenent de la tecnologia.
- **Independent de la interfície d'usuari**

  La interfície d'usuari pot modificar-se fàcilment sense tenir la necessitat de canviar la resta de components del sistema. Per exemple, una interfície web potser substituïda per una interfície de tipus consola sense canviar les regles de negoci o casos d'ús.

- **Independent de la base de dades**

  Es pot modificar el motor de base de dades per un altre de diferent sense modificar les regles de negoci.

- **Independent de qualsevol agent extern**

  De forma genèrica, la lògica de negoci no coneix ni té la responsabilitat de conèixer res del món exterior ni de la tecnologia emprada per a dur a terme els diferents propòsits.

7.1.2. Injecció de dependències

En l'apartat anterior, s'ha citat el concepte d'injecció de dependències. S'utilitza aquesta tècnica per a ***injectar els adapters secundaris a la lògica de negoci***.

Per poder descriure i reforçar aquest concepte se n'introduirà un altre anomenat Clean Architecture [31], creat per Robert C. Martin. A continuació es pot observar un esquema que il·lustra aquest tipus d'arquitectura:

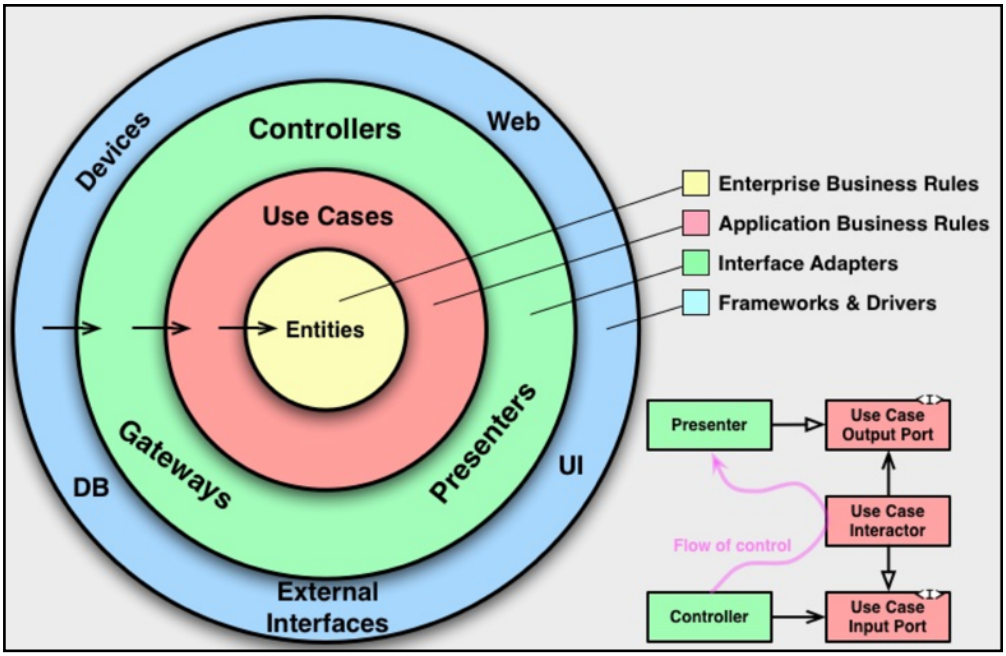


Figura 3 7.1.2 Injecció de dependències

Concepte molt proper al de l'hexagonal arquitecture i al de l'onion architecture [32] de Jeffrey Palermo.

El sistema es pot interpretar en cercles concèntrics, on tenim, del més interior al més exterior:

- el nucli del sistema o les regles de negoci (entitats)
- l'aplicació de les regles de negoci (casos d'ús)
- interfícies i adapters (controllers, presentadors)
- frameworks i drivers (bases de dades, web, dispositius, interfícies externes, interfícies d'usuaris)

D'aquesta forma, ***la capa més concèntrica és la més abstracta i és el nucli del sistema i la més externa és la més especifica i té el detall del sistema lligat a les tecnologies emprades***.

Entre les diferents capes que formen el sistema existeix una norma anomenada la **regla de la dependència** [33]. Els cercles concèntrics representen les diferents àrees del software. **En general com més cap a l'exterior són les capes, el nivell del software s'eleva, és a dir, es concreta i es torna més específic i pel contrari baixa l'abstracció**.

Es podria dir que **els cercles exteriors són mecanismes, el detall de *com s'han de realitzar les accions*** i els **més interiors són polítiques i contractes, expressen *què s'ha de fer***. Aquesta norma exposa que ***les dependències de codi només es poden donar cap als cercles dels més exteriors als interiors***. En cap cas es pot donar una dependència d'un cercle intern cap a un extern.

En particular el nom d'algun element declarat en una capa més externa no hauria d'estar mencionat en un element d'una capa més interna. Aquesta norma s'aplica a funcions, classes, variables o qualsevol altre tipus d'entitat o component software.

D'aquesta forma i per injectar les dependències externes cap a les capes més internes, **es creen les instàncies d'aquests objectes en la capa exterior i es passen com a paràmetres d'entrada a les funcions constructores dels objectes de les capes més internes**.

És a dir, si es vol injectar un objecte de tipus adapter secundari el qual implementa un port secundari cap a la lògica de negoci, es crea la

instància en la capa externa i se li injecta a la constructora com a port secundari de la lògica de negoci.

Cal recordar que els ports secundaris tan sols són interfícies que indiquen quines accions es poden realitzar, sense cap tipus de detall d'implementació. D'aquesta forma, es manté el desacoblament i l'abstracció en el sistema.

7.1.3. Capes d'arquitectura

Una vegada s'ha descrit de forma genèrica el concepte d'arquitectura hexagonal, en aquest apartat s'identifiquen les diferents capes que compondran l'arquitectura del sistema, així com els components que les capes contenen en el seu interior. Les capes estan representades en subdirectoris de fitxers ubicats en l'arrel del directori src.

Les capes són les següents:

- App
- Domain
- Interactor
- Infrastructure

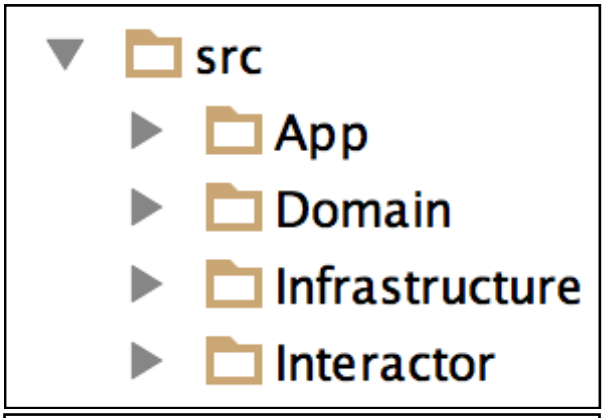


Figura 1 7.1.3 Estructura directoris

En la capa o directori **App** de l'aplicació es troben els adpters primaris. En aquesta primera fase del projecte, existeixen adapters primaris de tipus controladors web o de tipus consola. Aquests crearan els adapters secundaris en funció de les seves necessitats i invocaran els ports primaris per a dur a terme una funcionalitat concreta.

La capa **Domain** conté el nucli de l'aplicació. Els components d'aquesta capa són les entitats de domini necessàries per implementar la lògica de negoci i les interfícies dels ports secundaris que tan sols són un conjunt de mètodes que defineixen un contracte d'ús.

La capa **Infrastructure** conté els adapters secundaris. Aquests són la implementació dels mètodes dels ports secundaris de la capa Domain. S'ha de notar que per un mateix port secundari poden existir diferents adapters secundaris que implementin amb diferents tecnologies els mateixos mètodes que ofereix l'adapter.

Finalment, la capa **Interactor** conté l'aplicació de la lògica de negoci. Interactua amb la resta de capes de l'arquitectura. Els components principals implementen els casos d'ús. Aquests exposen els ports primaris als adapters primaris els quals estan ubicats a la capa App. També són els elements que reben per injecció de dependències els adapters secundaris. Utilitzaran els mètodes dels ports secundaris, els quals són els contractes preestablerts de quines funcionalitats pot invocar el cas d'ús dels adapters secundaris.

En els següents apartats es detallen els components de cada capa.

7.1.4. Domain

Es tracta del core o nucli de l'aplicació. En aquesta capa es troben els següents elements:

- **Entitats**

  Són components software els quals representen conceptes de la realitat. S'identifiquen i es modelen dins el context de gamification els elements necessaris per a poder aplicar la lògica de negoci necessària per assolir un propòsit.

- **Repositoris**

  Són els ports secundaris, interfícies que es composen per un conjunt de firmes de mètodes sense implementar.

- **Data Transformers**

Aquests components són necessaris per no exposar les entitats de domini a capes externes. En lloc d'aquestes, s'utilitzen uns components alternatius que creen els data transformers per exposar només la informació relativa a les entitats de domini sense la necessitat d'exposar les mateixes entitats.

- **Validator**

Encapsulen la lògica de validació de les entitats. Serveixen per a validar la correctesa dels camps i de la seva informació.

- **Exception**

Es tracten de les excepcions que es llencen en cas que el component validator detecti un error en la validació d'algun camp d'una entitat en concret.

- **Service**

Es tracten de components que compleixen amb una finalitat molt concreta necessària per aplicar lògica de negoci. Són com els repositoris, només consten d'una quantitat de firmes de mètodes els quals seran implementats per serveis d'infraestructura.

7.1.4.1. Entitats

Una entitat és per definició una **representació de la realitat d'un element en un context determinat**. Aquest element té atributs els quals els seus valors el fa únic. Durant el procés d'especificació, s'ha analitzat el context de gamification i s'han identificat les entitats del nostre model així com la informació que han de contenir cadascuna d'elles. Les entitats així com els seus components associats (Validators, Exceptions, Repositoris i Data Transformers) es troben ubicades a la capa Domain al subdirectori Entity.

7.1.4.1.1. Badge

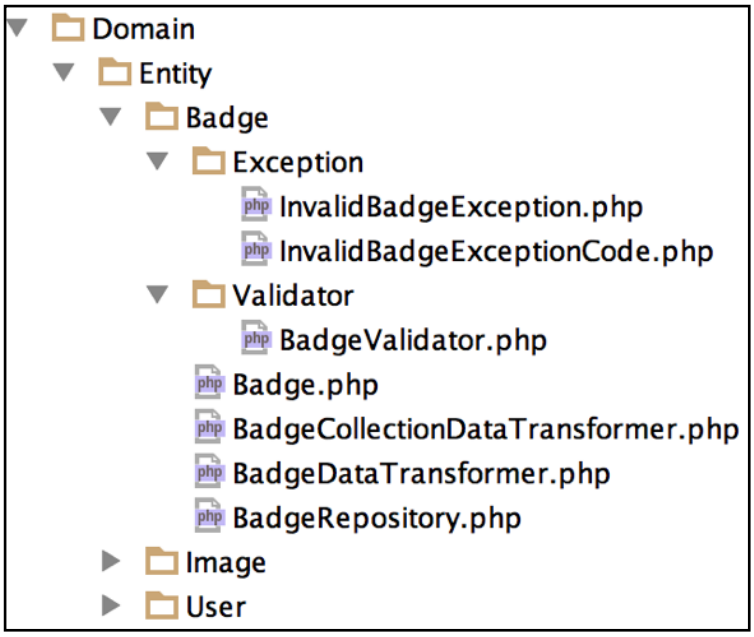


Figura 1 7.1.4.1.1 Estructura directoris Badge

L'entitat *Badge* representa el concepte Badge en el context de gamification. Es tracta de l'incentiu que se li dóna a l'usuari pel fet d'haver aconseguit un objectiu.

Aquesta entitat consta dels següents atributs:

```php
class Badge
{
    /** @var  string */
    private $id;
    /** @var  string */
    private $name;
    /** @var  string */
    private $description;
    /** @var  bool */
    private $isMultiUser;
    /** @var  User */
    private $user;
    /** @var  Image */
    private $image;
```

Figura 2 7.1.4.1.1 Badge Atributs

- **id**
  L'identificador de l'entitat. És un valor de tipus alfanumèric el qual és únic per a cada identitat.
- **name**
  El nom del badge. És de tipus text.
- **description**
  La descripció del badge. Serveix per a indicar informació tal com el tipus o instruccions de com es pot aconseguir el badge. És de tipus text.
- **isMultiUser**
  Informa si el badge és privat, és a dir només accessible per l'usuari del sistema que l'ha creat o pel contrari és públic i accessible per a tots els usuaris del sistema. És de tipus booleà. En el cas que el valor sigui veritat, el badge és públic. Altrament si el seu valor és fals el badge és privat.
- **user**
  L'usuari del sistema que ha creat el badge. És de tipus entitat User i conté tota la informació relativa a l'usuari del sistema.
- **image**
  La imatge associada al badge. És de tipus entitat Image i conté tota la informació relativa a la imatge del badge.

Tots els atributs són obligatoris amb excepció del cas de la descripció la qual es pot informar amb un text buit. El mètode constructor de la classe conté els valors de tots els atributs per paràmetre d'entrada. Aquest invoca els mètodes setters de cada atribut els quals modifiquen els valors dels atributs de l'entitat. Una vegada s'han modificat tots els valors, s'invoca el mètode validate *per comprovar la correctesa de tipus i de contingut de cadascun dels atributs*.

```php
public function __construct(
    $id,
    $name,
    $description,
    $isMultiUser,
    User $user,
    Image $image
) {
    $this->setId($id)
         ->setName($name)
         ->setDescription($description)
         ->setIsMultiUser($isMultiUser)
         ->setUser($user)
         ->setImage($image)
         ->validate();
}
```

Figura 3 7.1.4.1.1 Badge mètode constructor

El mètode validate crearà una classe de tipus *BadgeValidator* injectant-li l'entitat *Badge* i invocarà el mètode validate del validator. D'aquesta forma es crea un agregat del validator amb l'entitat la qual aquest valida. En el cas que l'entitat s'elimini del sistema per la raó que sigui, el seu validator també ha de deixar d'existir, ja que no té sentit la seva existència si no existeix l'entitat que ha de validar.

```php
/**
 * @return Badge
 */
private function validate()
{
    $this->buildValidator()->validate();

    return $this;
}

/**
 * @return BadgeValidator
 */
private function buildValidator()
{
    return new BadgeValidator($this);
}
```

Figura 4 7.1.4.1.1 Badge mètode validate

El mètode validate de la classe *BadgeValidator* té com a objectiu validar la correctessa tant de tipus com de contingut de l'entitat *Badge*.

```
public function validate()
{
    $this->validateId()
         ->validateName()
         ->validateDescription()
         ->validateIsMultiUser();
}
```

Figura 5 7.1.4.1.1 BadgeValidator validate

A continuació s'il·lustra la validació d'atributs mitjançant com a exemple l'atribut id de l'entitat *Badge*. La classe *BadgeValidator* té un mètode anomenat *validateId* el qual comprova que l'atribut id estigui informat (mètode *checkIdNotNull*) i que tingui el format correcte (mètode *checkIdFormat*). En tots els mètodes d'aquesta classe, si l'atribut no té un valor correcte, es llança una excepció de tipus *InvalidBadgeException* amb un codi d'excepció el qual indica quin paràmetre no conté el valor correcte. D'aquesta forma, el mètode *checkIdNotNull*, comprova que l'atribut tingui algun valor informat. Si no és així, es llença una excepció de tipus *InvalidBadgeException* ,creada a partir del mètode *buildInvalidBadgeException*, amb el codi d'excepció *STATUS_CODE_ID_NOT_PROVIDED* i s'aturà l'execució del programa. En cas que l'atribut tingui un valor assignat, es continua amb la resta de validacions d'atributs. En cas que tots els atributs compleixin amb les condicions de correctesa, es crearà l'entitat.

```
/**
 * @param int $statusCode
 *
 * @return InvalidBadgeException
 */
private function buildInvalidBadgeException($statusCode)
{
    return new InvalidBadgeException($statusCode);
}
```

Figura 6 7.1.4.1.1 BadgeValidator mètode buildInvalidBadgeException

```php
/**
 * @return BadgeValidator
 * @throws InvalidBadgeException
 */
private function validateId()
{
    $this->checkIdNotNull()
         ->checkIdFormat();

    return $this;
}

/**
 * @return BadgeValidator
 * @throws InvalidBadgeException
 */
private function checkIdNotNull()
{
    $aNullId = null;
    if ($this->badge->id() === $aNullId) {
        throw $this->buildInvalidBadgeException(
            InvalidBadgeExceptionCode::STATUS_CODE_ID_NOT_PROVIDED
        );
    }

    return $this;
}
```

Figura 7 7.1.4.1.1 BadgeValidator mètodes validateId i checkIdNotNull

7.1.4.1.2. User

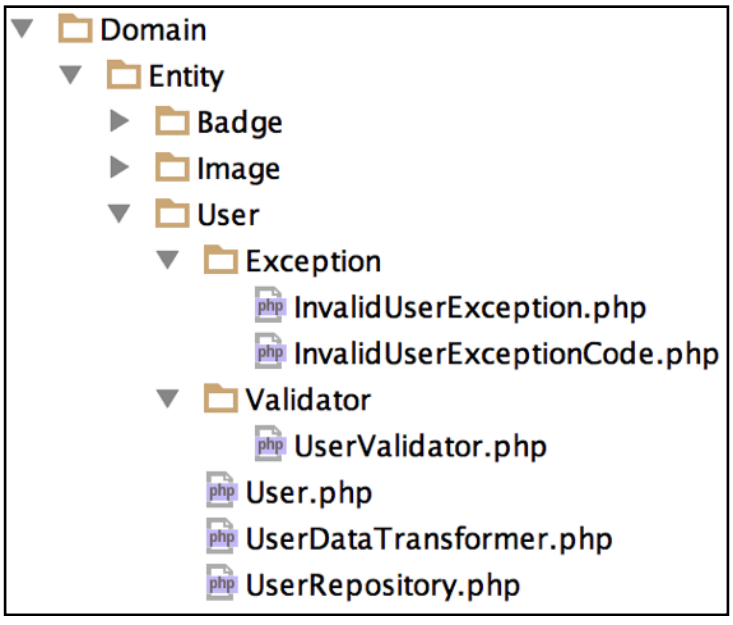


Figura 1 7.1.4.1.2 Estructura directoris Badge

L'entitat *User* representa l'usuari del sistema en el context del projecte. **S'ha de notar la diferència entre l'usuari del sistema en el context del projecte i l'usuari en el context de gamification**. El primer és el que utilitza el sistema per a gestionar els badges, és a dir, crear, modificar o eliminar badges. El segon és l'usuari que exerceix l'activitat de gamification al qual se li poden assignar badges en funció dels objectius que aconsegueix.

Aquesta entitat consta dels següents atributs:

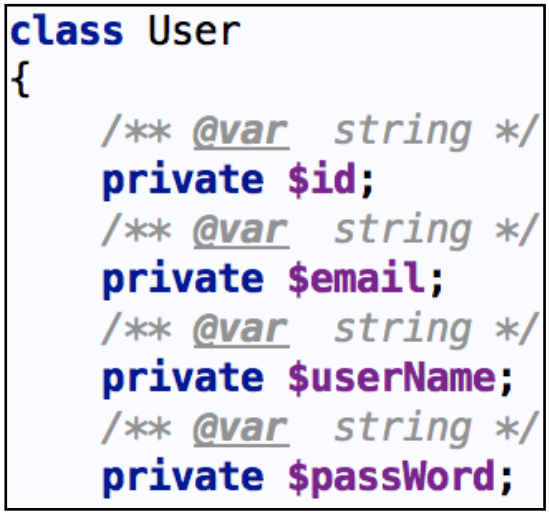


Figura 2 7.1.4.1.2 User Atributs

- **id**
  L'identificador de l'entitat. És un valor de tipus alfanumèric el qual és únic per a cada entitat.
- **email**
  El correu electrònic de l'usuari. És un valor de tipus text amb format estàndard de email el qual és únic per a cada entitat.
- **userName**
  El nom d'usuari utilitzat durant la fase de login. És un valor de tipus text el qual és únic per a cada entitat.
- **passWord**
  La contrasenya la qual és utilitzada conjuntament amb el username durant la fase de login. És un valor de tipus alfanumèric.

Tots els atributs són obligatoris. El mètode constructor de la classe conté els valors de tots els atributs per paràmetres. Aquest invoca els mètodes setters de cada atribut els quals modifiquen els valors dels atributs de l'entitat. Una vegada s'han modificat tots els valors, s'invoca el mètode validate *per comprovar la correctesa de tipus i de contingut de cadascun dels atributs*.

```
public function __construct(
    $id,
    $email,
    $userName,
    $passWord
) {
    $this->setId($id)
         ->setEmail($email)
         ->setUserName($userName)
         ->setPassWord($passWord)
         ->validate();
}
```

Figura 3 7.1.4.1.2 User mètode constructor

El mètode validate crearà una classe de tipus *UserValidator* injectant-li l'entitat *User* i invocarà el mètode validate del validator. D'aquesta forma es crea un agregat del validator amb l'entitat la qual aquest valida. En el cas que l'entitat s'elimini del sistema per la raó que sigui, el seu validator també ha de deixar d'existir, ja que no té sentit la seva existència si no existeix l'entitat que ha de validar.

```php
/**
 * @return User
 */
private function validate()
{
    $this->buildValidator()->validate();

    return $this;
}

/**
 * @return UserValidator
 */
private function buildValidator()
{
    return new UserValidator($this);
}
```

Figura 4 7.1.4.1.2 User mètode validate

El mètode validate de la classe *UserValidator* té com a objectiu validar la correctessa tant de tipus com de contingut de l'entitat *User*.

```php
public function validate()
{
    $this->validateId()
         ->validateEmail()
         ->validateUserName()
         ->validatePassWord();
}
```

Figura 5 7.1.4.1.2 UserValidator mètode validate

A continuació s'il·lustra la validació d'atributs mitjançant com a exemple l'atribut id de l'entitat *User*. La classe *UserValidator* té un mètode anomenat *validateId* el qual comprova que l'atribut id estigui informat (mètode *checkIdNotNull*) i que tingui el format correcte (mètode *checkIdFormat*). En tots els mètodes d'aquesta classe, si l'atribut no té un valor correcte, es llança una excepció de tipus *InvalidUserException* amb un codi d'excepció el qual indica quin paràmetre no conté el valor correcte.

D'aquesta forma, el mètode *checkIdNotNull*, comprova que l'atribut tingui algun valor informat. Si no és així, es llença una excepció de tipus *InvalidUserException* ,creada a partir del mètode *buildInvalidUserException*, amb el codi d'excepció *STATUS_CODE_ID_NOT_PROVIDED* i s'aturà l'execució del programa. En cas que l'atribut tingui un valor assignat, es continua amb la resta de validacions d'atributs. En cas que tots els atributs compleixin amb les condicions de correctesa, es crearà l'entitat.

```php
/**
 * @return UserValidator
 * @throws InvalidUserException
 */
private function validateId()
{
    $this->checkIdNotNull()
         ->checkIdFormat();

    return $this;
}

/**
 * @return UserValidator
 * @throws InvalidUserException
 */
private function checkIdNotNull()
{
    $aNullId = null;
    if ($aNullId === $this->user->id()) {
        throw $this->buildInvalidUserException(
            InvalidUserExceptionCode::STATUS_CODE_ID_NOT_PROVIDED
        );
    }

    return $this;
}
```

Figura 6 7.1.4.1.2 UserValidator mètode validateId i checkidNotNull

```php
/**
 * @param int $statusCode
 *
 * @return InvalidUserException
 */
private function buildInvalidUserException($statusCode)
{
    return new InvalidUserException($statusCode);
}
```

Figura 7 7.1.4.1.2 UserValidator mètode buildInvalidException

7.1.4.1.3. Image

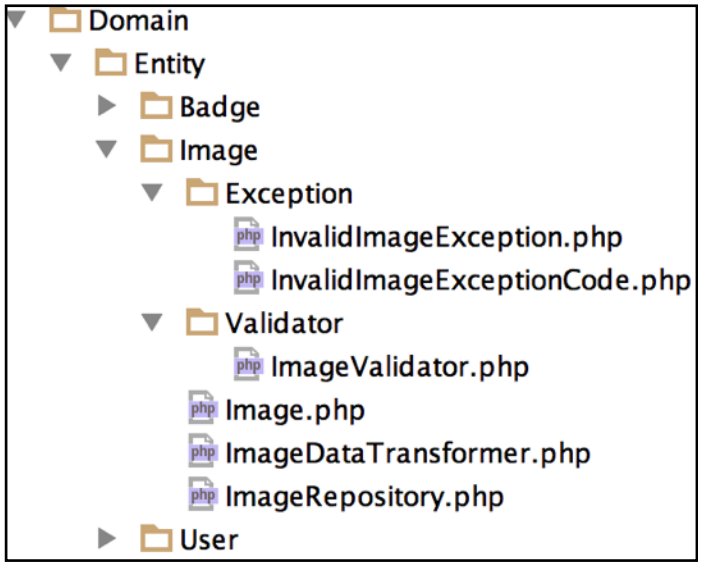


Figura 1 7.1.4.1.3 Estructura directoris Image

L'entitat *Image* representa la imatge associada a l'entitat Badge. És la representació gràfica del badge.

Aquesta entitat consta dels següents atributs:

```
class Image
{
    /** @var  string */
    private $id;
    /** @var  string */
    private $name;
    /** @var  int */
    private $width;
    /** @var  int */
    private $height;
    /** @var  string */
    private $format;
```

Figura 2 7.1.4.1.3 Image atributs

- **id**
  L'identificador de l'entitat. És un valor de tipus alfanumèric el qual és únic per a cada entitat.
- **name**
  El nom de la imatge. És un valor de tipus text.
- **width**
  L'amplada de la imatge. És un valor de tipus numèric més gran que 0.
- **height**
  L'alçada de la imatge. Junt amb l'amplada, defineix les dimensions de la imatge. És un valor de tipus numèric més gran que 0.
- **format**
  Es tracta del tipus de format de la imatge. És l'extensió de fitxer de la imatge, per exemple, imatge.jpeg. És un valor de tipus text i ha de coincidir amb algun dels següents valors: jpeg, tiff, gif, bmp, png.

Tots els atributs són obligatoris. El mètode constructor de la classe conté els valors de tots els atributs per paràmetres. Aquest invoca els mètodes setters de cada atribut els quals modifiquen els valors dels atributs de l'entitat. Una vegada s'han modificat tots els valors, s'invoca el mètode validate *per comprovar la correctesa de tipus i de contingut de cadascun dels atributs*.

```php
public function __construct(
    $id,
    $name,
    $width,
    $height,
    $format
) {
    $this->setId($id)
         ->setName($name)
         ->setWidth($width)
         ->setHeight($height)
         ->setFormat($format)
         ->validate();
}
```

Figura 3 7.1.4.1.3 Image mètode constructor

El mètode validate crearà una classe de tipus *ImageValidator* injectant-li l'entitat *Image* i invocarà el mètode validate del validator. D'aquesta forma es crea un agregat del validator amb l'entitat la qual aquest valida. En el cas que l'entitat s'elimini del sistema per la raó que sigui, el seu validator també ha de deixar d'existir, ja que no té sentit la seva existència si no existeix l'entitat que ha de validar.

```php
/**
 * @return Image
 */
private function validate()
{
    $this->buildValidator()->validate();

    return $this;
}

/**
 * @return ImageValidator
 */
private function buildValidator()
{
    return new ImageValidator($this);
}
```

Figura 4 7.1.4.1.3 Image mètode validate

El mètode validate de la classe *ImageValidator* té com a objectiu validar la correctesa tant de tipus com de contingut de l'entitat *Image*.

```
public function validate()
{
    $this->validateId()
         ->validateName()
         ->validateWidth()
         ->validateHeight()
         ->validateFormat();
}
```

Figura 5 7.1.4.1.3 ImageValidator mètode validate

A continuació s'il·lustra la validació d'atributs mitjançant com a exemple l'atribut id de l'entitat *Image*. La classe *ImageValidator* té un mètode anomenat *validateId* el qual comprova que l'atribut id estigui informat (mètode *checkIdNotNull*) i que tingui el format correcte (mètode *checkIdFormat*). En tots els mètodes d'aquesta classe, si l'atribut no té un valor correcte, es llança una excepció de tipus *InvalidImageException* amb un codi d'excepció el qual indica quin paràmetre no conté el valor correcte.

D'aquesta forma, el mètode *checkIdNotNull*, comprova que l'atribut tingui algun valor informat. Si no és així, es llença una excepció de tipus *InvalidImageException* ,creada a partir del mètode *buildInvalidImageException*, amb el codi d'excepció *STATUS_CODE_ID_NOT_PROVIDED* i s'aturà l'execució del programa. En cas que l'atribut tingui un valor assignat, es continua amb la resta de validacions d'atributs. En cas que tots els atributs compleixin amb les condicions de correctessa, es crearà l'entitat.

```php
/**
 * @throws InvalidImageException
 */
private function validateId()
{
    $this->checkIdIsNotNull()
         ->checkIdFormat();

    return $this;
}

/**
 * @return ImageValidator
 * @throws InvalidImageException
 */
private function checkIdIsNotNull()
{
    $aNullId = null;
    if ($this->image->id() === $aNullId) {
        throw $this->buildInvalidImageException(
            InvalidImageExceptionCode::STATUS_CODE_ID_NOT_PROVIDED
        );
    }

    return $this;
}
```

Figura 6 7.1.4.1.3 ImageValidator mètode validateId i checkIdIsNotNull

```php
/**
 * @param int $statusCode
 *
 * @return InvalidImageException
 */
private function buildInvalidImageException($statusCode)
{
    return new InvalidImageException($statusCode);
}
```

Figura 7 7.1.4.1.3 ImageValidator mètode buildInvalidImageException

7.1.4.1.4. Conceptes bàsics Domain Driven Design

Domain Driven Design (DDD) és un enfoc per al desenvolupament de software amb necessitats complexes mitjançant una profunda connexió entre la implementació i els conceptes del model amb la lògica de negoci. Es tracta d'un concepte molt ampli introduït per Eric Evans [34]. En aquest apartat s'expliquen els conceptes bàsics de Domain Driven Design aplicats a la capa de domini.

En primer lloc aquest tipus d'enfoc contraposa els **conceptes d'un domini anèmic amb el d'un domini ric**. El primer és aquell el qual el software que implementa les entitats té molt poca o no té lògica de negoci. En aquest context, les entitats de domini es redueixen a uns objectes que tan sols contenen informació però que aquesta informació ha estat processada i validada des d'un agent exterior, on la pròpia entitat, no ha tingut l'oportunitat de participar en el seu processament o validació. Pel que fa al domini ric és aquell en què les entitats tenen la lògica necessària no tan sols per emmagatzemar informació sinó també per realitzar les validacions necessàries així com el processament de la informació. D'aquesta forma, se li proporciona responsabilitat a les entitats per a la gestió de la informació que contenen. El processament i validació es realitza en un únic punt i no es distribueix en la resta de codi el qual utilitza aquesta entitat satisfent, d'aquesta forma, la definició estricte d'encapsulament.

Per a construir un model de domini ric, s'han pres tres decisions en el procés d'implementació de les entitats:

- **El mètode constructor conté tota la informació**
  L'agent extern que crea una entitat ha d'informar tota les dades com a paràmetres d'entrada en el mètode constructor. És la seva responsabilitat conèixer quina informació és necessària per a crear l'entitat.
- **Mètodes setters d'atributs amb visibilitat privada**
  Relacionat amb el punt anterior i una vegada passada tota la informació com a paràmetres d'entrada del mètode constructor, no hi ha necessitat d'exposar el mètodes setters a l'exterior. D'aquesta forma es modifica la visibilitat d'aquests mètodes per a què siguin privats i només puguin ser invocats des de la mateixa entitat. Aquest fet evita que durant el flux del programa es puguin modificar els atributs de les entitats un cop creades minimitzant les inconsistències de dades.
- **Les entitats es validen elles mateixes**
  Tal com s'ha explicat en la descripció de les entitats en els apartats anteriors, les entitats tenen un validator associat que conté tota la lògica de validació. Això és així perquè l'entitat té la responsabilitat de

```php
/**
 * @param string $id
 *
 * @return Badge
 */
private function setId($id)
{
    $this->id = $id;

    return $this;
}
```

Figura 1 7.1.4.1.4 Mètode setter privat

validar-se ella mateixa. Per això el validator és un agregat de l'entitat i no pot existir si no existeix l'entitat que valida.

7.1.4.2. Repositoris

Els repositoris de la capa de domini són els *ports secundaris* de l'arquitectura hexagonal. La seva **funció bàsica és la gestió d'entitats de la capa de domini**, és a dir, la seva persistència, modificació, esborrat o obtenció donats una serie de paràmetres. Són **interfícies que agrupen un conjunt de firmes de mètodes les quals no tenen implementació definida**. D'aquesta forma es defineix un contracte dels mètodes que es poden invocar per a gestionar les entitats però sense definir com es realitzen aquestes operacions. El detall de la implementació es realitzarà a la capa infrastructure.

Es troben ubicats en l'arquitectura en el mateix directori de l'entitat que gestionen.

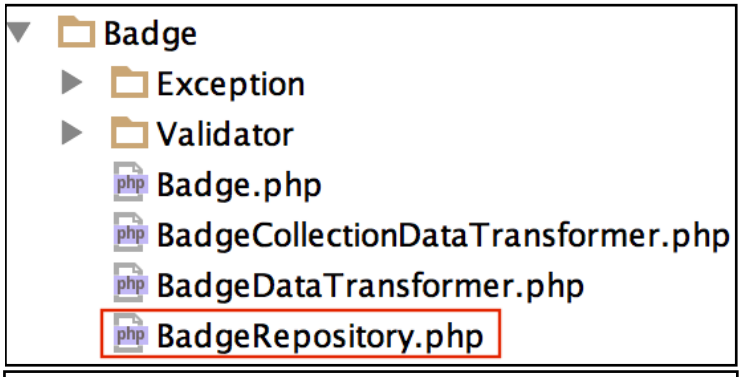


Figura 2 7.1.4.2 Localització BadgeRepository

A continuació es pot observar un exemple de codi del BadgeRespository.

```php
interface BadgeRepository
{
    /**
     * @param Badge $badge
     */
    public function persist(Badge $badge);

    /**
     * @param string $id
     *
     * @return Badge | null
     */
    public function find($id);

    /**
     * @param Badge $badge
     */
    public function remove(Badge $badge);

    /**
     * @param User $user
     *
     * @return Badge[]
     */
    public function findByUser(User $user);

    /**
     * @return Badge[]
     */
    public function findMultiUser();
}
```

Figura 3 7.1.4.2 BadgeRepository interface

Com es pot observar, es tracta d'una interfície amb les següents signatures de mètodes:

- ***persist***
  Donada una entitat Badge la registra en el sistema.
- ***find***
  Donat un id retorna una entitat de tipus Badge o null si no existeix.
- ***remove***
  Donada una entitat Badge l'elimina del sistema.

- ***findByUser***
  Donada una entitat User retorna tots les entitats Badges relacionades amb l'entitat User.
- ***findMultiUser***
  Retorna totes les entitats Badge que tenen l'atribut isMultiUser al valor true.

La lògica de cada mètode serà implementada pels adapters secundaris o repositoris de la capa d'infraestructura.

7.1.4.3. Data Transformers

Els **DataTransformers són uns components software que permeten ocultar la lògica de les entitats de domini a la resta de capes**. En algunes ocasions, existeix la necessitat per part de capes externes de consultar la informació que conté una entitat. Aquests components transformen les entitats en un altre tipus d'objecte per evitar exposar les entitats a capes més externes.

Semblant als repositoris, en la capa de domini els data transformers són una interfície que conté la signatura dels mètodes que es poden utilitzar. La seva implementació es realitzarà per elements de la capa d'infraestructura.

Com a extensió dels DataTransformers, existeixen els CollectionDataTransformers els quals transformen un conjunt d'entitats.

Aquests components estan ubicats en la capa de domini dins de cada directori de cada entitat. A continuació es mostren la localització els DataTransformers associats a l'entitat Badge i el seu codi software.

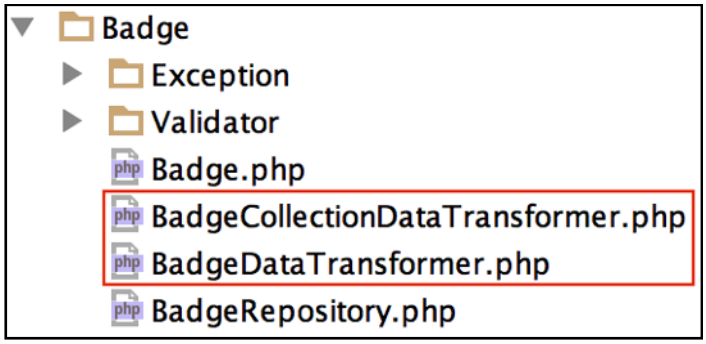


Figura 1 7.1.4.3 DataTransformers entitat Badge

```php
interface BadgeDataTransformer
{
    /**
     * @param Badge $badge
     *
     * @return mixed
     */
    public function transform(Badge $badge);
}
```

Figura 2 7.1.4.3 BadgeDataTransformer interface

```php
interface BadgeCollectionDataTransformer
{
    /**
     * @param Badge[] $badges
     *
     * @return mixed
     */
    public function transform($badges);
}
```

Figura 3 7.1.4.3 BadgeCollectionDataTransformer interface

Les dues interfícies consten d'un mètode *transform* que reben com a paràmetre d'entrada una o una col·lecció d'entitats.

Aquests components són especialment útils en el cas que s'hagi de modificar el nom d'algun mètode d'una entitat. És necessari modificar el nom del mètode en la implementació dels DataTransformers per a consultar la informació que conté l'entitat, però aquest canvi potser transparent a la resta de capes **facilitant, d'aquesta forma, la modificació o refactorització de codi**.

7.1.4.4. Serveis

Els **Serveis de domini solen ser components software que realitzen funcionalitats molt concretes les quals és solen reaprofitar en diferents punts del codi**, generalment injectats com a dependència als components de la capa interactor.

En aquesta fase inicial del projecte, els serveis de domini són semblants als repositoris i els data transformers. Són unes interfícies que defineixen una serie de mètodes els quals seran implementats pels components de la capa infrastructure.

Es troben ubicats a la capa de domini, dins del directori Service.

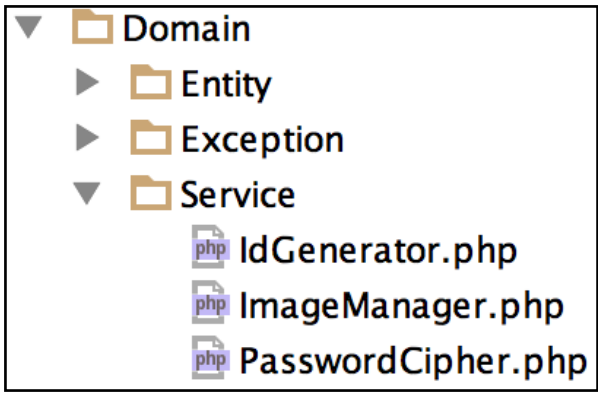


Figura 1 7.1.4.4 Domain Services

A continuació s'explica breument cada servei en què consisteix i es mostra el seu codi associat.

- ***IdGenerator***

  Es tracta d'un servei que genera ids únics.

```php
interface IdGenerator
{
    /**
     * @return string
     */
    public function generateId();
}
```

Figura 2 7.1.4.4 Servei IdGenerator

- ***PasswordCipher***

  Aquest servei encripta passwords.

```php
interface PasswordCipher
{
    /**
     * @param string $password
     *
     * @return string
     */
    public function cipher($password);
}
```

Figura 3 7.1.4.4 Servei PasswordCipher

- ***ImageManager***
  S'utilitza per gestionar les images físiques

```php
interface ImageManager
{
    /**
     * @param string $toPath
     * @param string $id
     * @param string $format
     */
    public function upload($toPath, $id, $format);

    /**
     * @param string $id
     * @param string $format
     */
    public function buildPath($id, $format);

    /**
     * @param string $id
     * @param string $format
     */
    public function remove($id, $format);
}
```

Figura 4 7.1.4.4 Sevei ImageManager

7.1.4.5. Tests Unitaris

Un **test unitari o prova unitària és una forma de comprovar el funcionament correcte d'un mòdul o una classe**. Serveix per assegurar que cadascun dels mòduls implementats funciona correctament per separat sense interaccionar amb la resta [35]. Els tests unitaris **solen ser automatitzats i garanteixen que, com a mínim una vegada, s'ha executat el codi implementat**. D'aquesta forma es minimitzen les possibilitats que existeixin errors en el codi. En aquest projecte s'ha utilitzat el *phpunit framework* [36] per implementar els tests unitaris.

S'ha pres la decisió de testejar de forma unitària els components de la capa Domain i de la capa Interactor. Aquestes dues capes formen el nucli del sistema on s'acumula tota la lògica de negoci la qual ha de garantir que funciona correctament.

En aquesta primera fase del projecte **s'ha descartat testejar de forma unitària les capes d'App i de Infrastructure,** ja que són

les que estan més lligades a la tecnologia. Quan els components contenen molt detall d'implementació tecnològica, els test unitaris solen ser molt fràgils dificultant el seu manteniment i solen aportar poc valor. En general, per testejar la tecnologia que utilitza el sistema, se sol fer amb *tests d'integració i de comportament*, els quals s'han plantejat desenvolupar en següents fases del projecte [36].

Bàsicament en aquesta capa **s'han testejat de forma unitària les entitats de domini y els seus validators associats**. En aquesta capa es té el **100% de coverage** de les classes testejades. El *coverage* [37] ens indica quin percentatge de línies de codi han estat executades pel test. En aquest cas s'han creat test per a què s'executin totes les línies de codi dels components software que formen part de la capa Domain.

Els tests estan ubicats dins el directori src/Test en el directori Domain. Dins d'aquest directori hi ha els directoris de les entitats que hi ha en la capa Domain. El nom dels tests tenen el nom de la classe amb el sufix test. Per exemple, el test del Badge s'anomena BadgeTest.

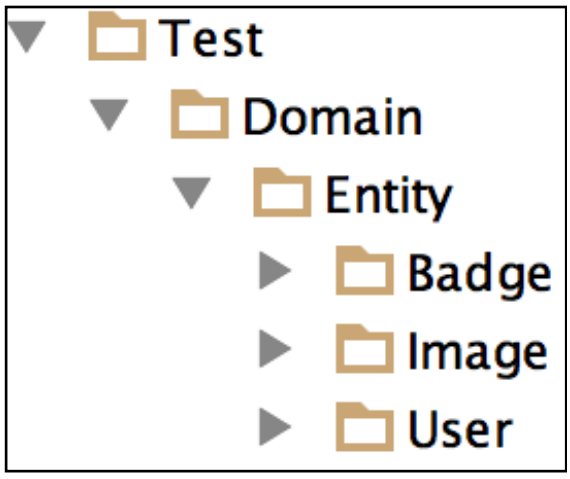
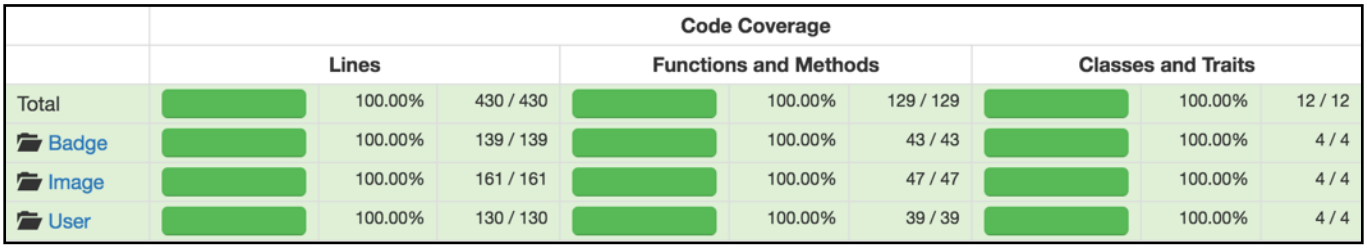


Figura 1 7.1.4.5 Domain test

En total s'han testejat 1855 línies de codi de les quals 436 (23,50%) són de la capa de domini, 537 mètodes dels quals 133 (24,78%) són de la capa de domini i 80 classes de les quals 13 (16,25%) són de la capa de domini.

	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total		100.00%	1855 / 1855		100.00%	537 / 537		100.00%	80 / 80
Domain		100.00%	436 / 436		100.00%	133 / 133		100.00%	13 / 13
Interactor		100.00%	1419 / 1419		100.00%	404 / 404		100.00%	67 / 67

Figura 2 7.1.4.5 Domain Code Coverage



	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total		100.00%	430 / 430		100.00%	129 / 129		100.00%	12 / 12
Badge		100.00%	139 / 139		100.00%	43 / 43		100.00%	4 / 4
Image		100.00%	161 / 161		100.00%	47 / 47		100.00%	4 / 4
User		100.00%	130 / 130		100.00%	39 / 39		100.00%	4 / 4

Figura 3 7.1.4.5 Entities Code Coverage

7.1.5. Interactor

És el **nucli o core del sistema**. Els components que formen part d'*aquesta capa interaccionen amb els components de la resta de capes* per a poder dur a terme funcionalitats concretes. Aquesta capa **conté la major part de lògica de negoci** la qual es distribueix exclusivament entre les capes domain i interactor. La **lògica de negoci determina com les dades d'entrada han de ser transformades o processades i quina informació s'ha de comunicar com a resultat de sortida** [39]. Implementa les regles de negoci del món real sense tenir en compte els detalls d'implementació, és a dir, com la informació és persistida o com s'ha de presentar. D'aquesta forma, les regles de negoci són independents de la interacció de l'usuari amb el sistema.

La lògica de negoci **s'ha identificat durant el procés despecificació i de disseny del projecte en forma de casos d'ús**. Els components d'aquesta capa implementaran en forma de codi software la definició dels diagrames de flux de cada cas d'ús els quals defineixen en detall totes les accions que s'han de realitzar per a assolir un objectiu determinat.

Un dels requisits desitjables en la implementació dels components d'aquesta capa és que el **codi software escrit tingui un alt nivell semàntic**. L'escenari ideal és que *una persona que no tingui coneixements tècnics però que sigui capaç d'entendre la lògica de negoci pugui ser capaç d'interpretar el codi dels components d'aquesta capa sense*

gaire dificultat. Per poder assolir aquest objectiu, els noms dels mètodes com el de les variables i l'ordre del flux del codi haurien de ser prou autoexplicatius i intuïtius per facilitar la seva comprensió.

La capa està ubicada dins els directori src. Al seu interior té un directori anomenat CommandHandler que conté tots els casos d'ús implementats actualment.

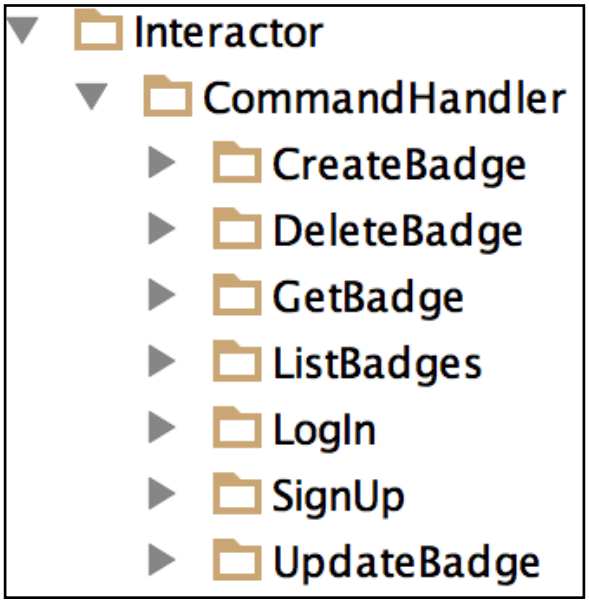


Figura 1 7.1.5. Llista Casos d'Ús

7.1.5.1. Command

Un Command és un component software el qual **contindrà tota la informació necessària per a què el command handler associat pugui realitzar una funció determinada**. L'objecte command és el paràmetre d'entrada de l'únic mètode públic del command handler.

La seva ubicació dins la capa Interactor és en el directori CommandHandler. Dins d'aquest directori existeixen directoris per a cada cas d'ús associats i dins d'aquests, es troben els command i command handler que implementen el cas d'ús.

Figura 1 7.1.5.1 ListBadges command i command handler

A continuació, s'il·lustra amb l'exemple *ListBadgesCommand* el funcionament general d'aquests tipus de components.

El command **rep com a paràmetres del mètode constructor totes les dades necessàries**. En aquest cas és el paràmetre $userId el qual es tracta d'un identificador d'un usuari del sistema. Actualment tots els paràmetres dels mètodes constructors dels commands són de tipus bàsic (cadenes de text, enters, booleans). **La informació està relacionada amb les dades necessàries per executar de forma estricta la lògica de negoci implementada en el command handler. En cap concepte, la informació del command contindrà informació relacionada amb la resta de capes que no siguin la capa Interactor**.

```php
class ListBadgesCommand
{
    /** @var string */
    private $userId;

    /**
     * @param string $userId
     */
    public function __construct($userId)
    {
        $this->setUserId($userId)
             ->validate();
    }
```

Figura 2 7.1.5.1 ListBadgesCommand mètode constructor

Tot seguit a actualitzar el valor de l'atribut *userId*, s'invoca el mètode *validate*. Aquest mètode construirà un nou objecte de tipus *ListBadgesCommandValidator* i s'invocarà al mètode *validate* d'aquest objecte el qual conté tota la lògica de validació de paràmetres per a garantir la seva correctesa. El flux de validació és el mateix que s'ha explicat per les entitats de la capa Domain. També compleixen amb els tres principis de disseny de DDD aplicats a les entitats de domini: el mètode constructor conté tota la informació necessària, els mètodes setters d'atributs tenen visibilitat privada i els commands es validen ells mateixos.

```php
/**
 * @return ListBadgesCommand
 */
private function validate()
{
    $this->buildValidator()->validate();

    return $this;
}

/**
 * {@inheritdoc}
 */
private function buildValidator()
{
    return new ListBadgesCommandValidator($this);
}
```

Figura 3 7.1.5.1 ListBadgesCommand mètode validate

7.1.5.2. Command Handlers

Els Command Handlers són els **components software que contenen la lògica de negoci d'un cas d'ús en concret**. De forma genèrica, aquests components tenen un **mètode constructor** el qual *rep totes les dependències necessàries* per poder dur a terme el flux del cas d'ús. També tenen un **únic mètode públic anomenat handle** el qual rep *com a paràmetre un objecte de tipus command*. El command contindrà tota la informació necessària per poder realitzar l'execució del codi.

Per il·lustrar el funcionament d'aquests components s'analitza el *ListBadgesCommandHandler* el qual implementa el cas d'ús *ListBadges*. En el mètode constructor d'aquest command handler, es pot

observar que s'injecten com a dependències tres components: el *BadgeRepository*, el *UserRepository* i el *BadgeCollectionDataTransformer*.

```
public function __construct(
    BadgeRepository                 $badgeRepository,
    UserRepository                  $userRepository,
    BadgeCollectionDataTransformer $badgeCollectionDataTransformer
) {
    $this->badgeRepository                 = $badgeRepository;
    $this->userRepository                  = $userRepository;
    $this->badgeCollectionDataTransformer = $badgeCollectionDataTransformer;
}
```

Figura 1 7.1.5.2 ListBadgesCommandHandler mètode constructor

El mètode handle del ListBadgesCommandHandler conté tota la lògica de negoci per executar el cas d'ús. Es pot observar que es basa en tres mètodes privats de la classe:

- ***tryToFindBadgesByUser***
  Donat un userId el qual és informat pel *ListBadgesCommand* es cerca en el sistema totes les entitats de tipus *Badge* que tenen com a propietari l'*User* amb l'identificador en qüestió.
- ***tryToFindMultiUserBadges***
  Es cerca totes les entitats de tipus *Badge* que són de tots els usuaris, és a dir, que tenen el valor de l'atribut *isMultiUser* a true.
- ***mixBadgesResult***
  Es mesclen les entitats *Badges* obtingudes com a resultat de la invocació dels dos mètodes anteriors.

Finalment, s'invoca el mètode *transform* de l'objecte *BadgeCollectionDataTransformer* per transformar les entitats de tipus *Badge* en altres objectes que puguin ser exposats a capes externes.

```php
/**
 * @param ListBadgesCommand $command
 *
 * @return mixed
 */
public function handle($command)
{
    $badgesByUser    = $this->tryToFindBadgesByUser($command->userId());
    $badgesMultiUser = $this->tryToFindMultiUserBadges();
    $badgesMixed     = $this->mixBadgesResult($badgesByUser, $badgesMultiUser);

    return $this->badgeCollectionDataTransformer->transform($badgesMixed);
}
```

Figura 2 7.1.5.2 ListBadgesCommandHandler mètode handle

7.1.5.3. Injecció de dependències

Com es pot observar en l'apartat anterior, el mètode constructor de la classe ListBadgesCommandHandler rep per paràmetres els repositoris necessaris i el data transformer per no exposar les entitats a l'exterior. Aquest fet és **la materialització en codi de la injecció de dependències**.

```php
public function __construct(
    BadgeRepository                $badgeRepository,
    UserRepository                 $userRepository,
    BadgeCollectionDataTransformer $badgeCollectionDataTransformer
) {
    $this->badgeRepository                = $badgeRepository;
    $this->userRepository                 = $userRepository;
    $this->badgeCollectionDataTransformer = $badgeCollectionDataTransformer;
}
```

Figura 1 7.1.5.3 Mètode constructor injecció de dependències

Les classes de les dependències tenen els noms de les interfícies de la capa de domini. En general, els command handlers rebran com a dependències objectes que implementin les interfícies de la capa de domini, és a dir, els *adapters secundaris*. En el codi implementat en els command handlers, **s'invocaran als mètodes definits en les interfícies dels ports primaris els quals pertanyen a la capa de domini. En cap concepte es trobaran detalls d'implementació tecnològica en el codi dels command handlers**.

```php
/**
 * @param string $userId
 *
 * @return Badge[]
 * @throws InvalidListBadgesCommandHandlerException
 */
private function tryToFindBadgesByUser($userId)
{
    $user = $this->tryToFindUserByUserId($userId);
    try {
        $badgesByUser = $this->badgeRepository->findByUser($user);
    } catch (\Exception $exception) {
        throw $this->buildListBadgesCommandHandlerException(
            InvalidListBadgesCommandHandlerExceptionCode::STATUS_CODE_BADGES_NOT_FOUND
        );
    }

    return $badgesByUser;
}
```

Figura 2 7.1.5.3 Invocació mètode findByUser del BadgeRepository

Com es pot observar, tota la implementació tecnològica per a cercar Badges donat un usuari s'encapsularà en el adapter secundari de la capa d'infraestructura que implementi la interfície de l'adapter primari BadgeRepository.

Aquest fet fa que **el codi sigui més intel·ligible**. Una persona amb coneixements del model de negoci de l'empresa pot entendre fàcilment que significa cercar en el BadgeRepository entitats Badges associades a un usuari en concret. Sense tenir coneixements tècnics li seria més difícil entendre l'execució d'una query de tipus mysql que realitzi una sentència sql select sobre una taula en concret.

Finalment i des del punt de vista de la mantenibilitat, la **injecció de dependències ens permet modificar la tecnologia implementada per l'adapter secundari amb un cost de manteniment zero pel codi del command handler**. És a dir, modificar un adapter secundari implementat amb un tecnologia per un altre implementat amb una tecnologia totalment diferent, no s'hauria de modificar cap línia de codi del command handler, ja que tan sols invoca els mètodes oferts per les interfícies dels adapters primaris de la capa Domain.

7.1.5.4. Test Unitaris

Com s'ha explicat en l'apartat de Test Unitaris de la capa Domain, s'ha decidit realitzar tests unitaris de la capa Interactor perquè conté la major part de la lògica de negoci del sistema. La lògica de negoci, representa el nucli del sistema el qual ha de garantir al màxim el seu correcte funcionament. Els tests unitaris són les eines necessàries per assolir aquest objectiu.

En aquesta capa s'**han testejat tant els Command com els CommandHandlers**. Com en el cas de la capa Domain s'ha utilitzat el **framework phpunit** per a implementar els units tests. Totes les classes tenen un **100% de coverage**. Aquest fet significa que tots els tests creats en la seva conjunció executen totes les línies de codi implementades en aquesta capa.

Els tests estan ubicats dins el directori src/Test en el directori Interactor. Dins d'aquest directori hi ha els directoris dels casos d'ús que hi ha en la capa Interactor. El nom dels tests tenen el nom de la classe amb el sufix test. Per exemple, el test del CreateBadgeCommandHandler s'anomena CreateBadgeCommandHandlerTest.

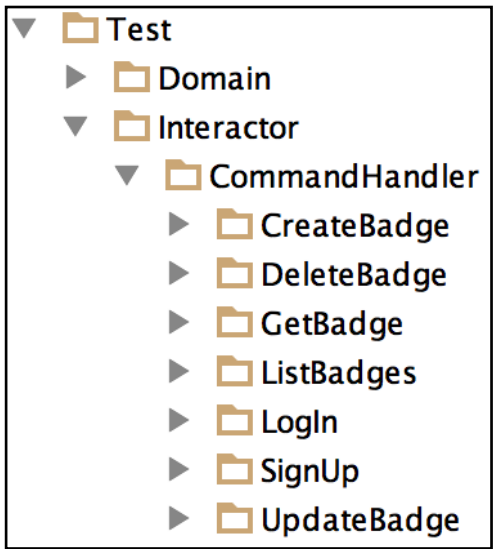


Figura 1 7.1.5.4 Interactor Test

En total s'han testejat 1855 línies de codi de les quals 1419 (76,50%) són de la capa interactor, 537 mètodes dels quals 404 (75,23%) són de la capa interactor i 80 classes de les quals 67 (83,75%) són de la capa interactor.

	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total		100.00%	1855 / 1855		100.00%	537 / 537		100.00%	80 / 80
Domain		100.00%	436 / 436		100.00%	133 / 133		100.00%	13 / 13
Interactor		100.00%	1419 / 1419		100.00%	404 / 404		100.00%	67 / 67

Figura 2 7.1.5.4 Interactor Code Coverage

	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total		100.00%	1413 / 1413		100.00%	400 / 400		100.00%	66 / 66
CreateBadge		100.00%	374 / 374		100.00%	109 / 109		100.00%	15 / 15
DeleteBadge		100.00%	129 / 129		100.00%	36 / 36		100.00%	7 / 7
GetBadge		100.00%	103 / 103		100.00%	31 / 31		100.00%	7 / 7
ListBadges		100.00%	91 / 91		100.00%	27 / 27		100.00%	7 / 7
LogIn		100.00%	112 / 112		100.00%	31 / 31		100.00%	7 / 7
SignUp		100.00%	161 / 161		100.00%	43 / 43		100.00%	7 / 7
UpdateBadge		100.00%	441 / 441		100.00%	122 / 122		100.00%	15 / 15

Figura 3 7.1.5.4 Use Case coverage

7.1.5.4.1. Test Driven Development

El test driven development o TDD [40] és *un mètode per desenvolupar software de forma iterativa el qual consisteix en la repetició de cicles* de desenvolupament molt curts anomenats baby steps.

En general, el cicle consisteix en els següents passos:

- En primer lloc, el programador escriu un test inicial el qual defineix una nova millora o funcionalitat que no existeix prèviament.
- El test fallarà en primera instància, ja que el codi que ha de fer que el test funcioni no està creat.
- En aquest punt el programador escriu el mínim codi possible per a què el test funcioni.
- El test no falla y garanteix la correctesa de la nova funcionalitat creada.
- En aquest punt, el programador refactoritza el codi que acaba de crear garantint que el test segueix funcionant.
- Es repeteix el cicle.

En aquest apartat es fa una descripció detallada d'aquest mètode i els objectius que es volen assolir amb la seva aplicació. S'il·lustra l'explicació amb un cas pràctic dut a terme durant el desenvolupament del projecte.

Fases i objectius

Kent Beck és un enginyer de software, creador del Extreme Programming [41] i un dels signataris del Agile Manifesto, conceptes molt lligats al TDD. En el seu llibre Test-Driven Development by Example [42], explica les diferents fases d'aquest mètode de desenvolupament.

Les següents premisses són bàsiques a l'hora d'aplicar aquesta metodologia:

- **No s'hauria d'escriure una nova línia de codi almenys que el programador hagi escrit primer un test que falli.**
- **Eliminar la duplicació de codi**

Algunes de les implicacions d'aquestes premisses són les següents:

- El disseny ha de ser orgànic

  És a dir, durant el procés s'ha de executar codi aportant feedback entre les diferents decisions preses.
- El propi programador ha d'escriure els tests

  D'aquesta forma es garantirà que no existeix codi sense test que el cobreixi i no depèn d'un tercer per a garantitr que el seu codi tingui els seus tests.
- Respostes ràpides a canvis petits

  El disseny hauria de ser altament cohesiu i desacoblat per a poder crear tests d'una forma senzilla. És a dir, molts elements no excessivament complexos interrelacionats entre si i sense fortes dependències ni acoblament.

A continuació, es descriuen les fases preestablertes o baby steps de la metodologia:

Vermell

Consisteix a escriure un petit test que no funciona, inclús que ni tan sols compila.

Verd

Fer el mínim codi possible i el més ràpid possible per a què el test funcioni.

Refactoritzar

Eliminar totes les duplicacions de codi identificades mantenint el test funcionant.

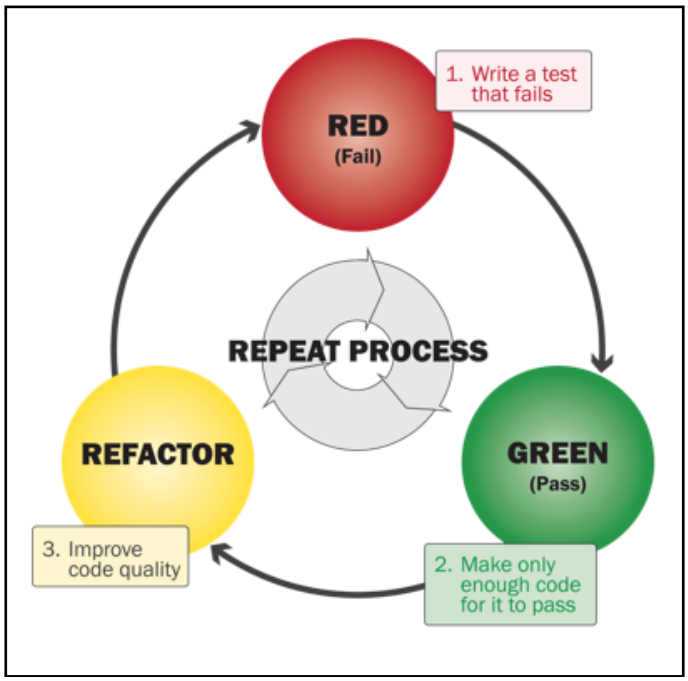


Figura 1 7.1.5.4.1 Metodologia TDD

Utilitzant aquesta metodologia de desenvolupament es persegueixen els següents objectius entre d'altres:

- No existeix codi sense el seu test associat

  D'aquesta forma la resta de programadors d'un projecte poden tenir les garanties i llibertat de modificar el codi mantenint la correctesa de la funcionalitat prèviament desenvolupada. En el cas que es modifiqui la funcionalitat anterior i no funcioni, els testos haurien de fallar.

- Reduir el nombre de bugs del sistema

  Aquest fet està altament relacionat amb l'eliminació de duplicació de codi. En el cas que existeix un error o bug en alguna línia de codi i aquesta està duplicada, s'ha de multiplicar l'error pel nombre de duplicitats.

- Codi net

  En la fase de refactor, a part d'eliminar la duplicitat de codi, també es pot aprofitar per utilitzar tècniques tals com extreure codi en mètodes, classe o constants fent d'aquesta. D'aquesta forma, se li atorga al codi d'intencionalitat i semàntica fent-lo més intel·ligible. Aquest fet, entre d'altres, es descriu de forma exhaustiva en el llibre Clean code: A Handbook of Agile Software Craftsmanship de l'autor Robert C. Martin [43].

TDD en el projecte

A l'inici del projecte, existeixen molts dubtes de com dissenyar els diferents elements els quals compondran el sistema i com interactuaran entre ells. Aquest fet implicava que totes les *decisions que s'han pres han estat sota un grau de feedback elevat i havien de ser constantment qüestionades*. Aquesta és la definició de **disseny orgànic**. Si en alguna part del procés del disseny implicava realitzar un canvi, ha suposat un fet clau i de gran ajuda a tenir el codi preexistent testejat. D'aquesta forma la fase de refactor s'ha pogut dur a terme amb totes les garanties necessàries.

S'ha aplicat TDD als components del sistema que s'han testejat, és a dir, al codi que es troba dins les capes Domain i Interactor. El codi que es troba ubicat en aquests directoris és el nucli de l'aplicació.

Exemple pràctic TDD

En aquest apartat es detalla un exemple real del projecte. D'aquesta forma s'il·lustrarà la teoria del TDD amb un exemple pràctic. L'exemple es realitza amb el codi de les classes *SignInCommand* i *SignInCommandTest*.

ESTAT INICIAL

En la classe *SignInCommandTest* existeix inicialment un test. El test té per nom *commandWithoutEmailShouldThrowExceptionWithEmailNotProvidedStatusCode*. El seu objectiu és comprovar que si no s'informa del camp *email* a la classe *SigInCommand*, aquesta classe haurà de llançar una excepció de tipus *InvalidCommandException* amb el codi d'excepció de tipus *STATUS_CODE_EMAIL_NOT_PROVIDED*. En qualsevol altre cas el test no funcionarà.

```php
use Interactor\CommandHandler\SignIn\Exception\InvalidCommandException;
use Interactor\CommandHandler\SignIn\SignInCommand;

class SignInCommandTest extends \PHPUnit_Framework_TestCase
{
    /**
     * @test
     */
    public function commandWithoutEmailShouldThrowExceptionWithEmailNotProvidedStatusCode()
    {
        try {
            $this->buildCommand(null);
            $this->thisTestFails();
        } catch (InvalidCommandException $invalidCommandException) {
            $this->assertEquals(
                InvalidCommandException::STATUS_CODE_EMAIL_NOT_PROVIDED,
                $invalidCommandException->code()
            );
        }
    }

    /**
     * @param string $email
     *
     * @return SignInCommand
     */
    private function buildCommand($email)
    {
        return new SignInCommand($email);
    }

    private function thisTestFails()
    {
        $this->assertTrue(false);
    }
}
```

Figura 2 7.1.5.4.1 Classe SignInCommandTest estat inicial

La classe *SignInCommand* consta d'un únic mètode *__construct* el qual permet crear noves instàncies de la classe i que rep com a paràmetre d'entrada el paràmetre *$email*. La seva implementació és simple

i l'únic que fa és crear una excepció de tipus *InvalidCommandException* el qual té com a paràmetre d'entrada el codi d'excepció *STATUS_CODE_EMAIL_NOT_PROVIDED* i la llança.

```php
use Interactor\CommandHandler\SignIn\Exception\InvalidCommandException;

class SignInCommand
{
    /**
     * @param string $email
     *
     * @throws InvalidCommandException
     */
    public function __construct($email)
    {
        throw new InvalidCommandException(
            InvalidCommandException::STATUS_CODE_EMAIL_NOT_PROVIDED
        );
    }
}
```

Figura 3 7.1.5.4.1 Classe SignInCommand estat inicial

La classe *InvalidCommandException* extend de la classe estàndard *Exception* de la llibreria nativa del llenguatge PHP. Es pot observar que consta d'un mètode *__construct* el qual s'utilitza per crear noves instàncies i té com a paràmetre d'entrada *$statusCode* el qual és el codi d'excepció. En el cas del codi *STATUS_CODE_EMAIL_NOT_PROVIDED* s'invoca el mètode privat *emailNotProvided* el qual modifica el codi d'execepció *STATUS_CODE_EMAIL_NOT_PROVIDED* utilitzant el mètode privat *setCode* de la mateixa classe. Finalment s'invoca al mètode *__construct* de la classe pare *Exception* informant del codi i el missatge d'excepció.

```php
class InvalidCommandException extends \Exception
{
    const STATUS_CODE_EMAIL_NOT_PROVIDED       = -1;

    /**
     * @param int $statusCode
     */
    public function __construct($statusCode)
    {
        switch ($statusCode) {
            case static::STATUS_CODE_EMAIL_NOT_PROVIDED:
                $this->emailNotProvided();
                break;
        }

        parent::__construct($this->message(), $this->code());
    }

    /**
     * @return InvalidCommandException
     */
    private function emailNotProvided()
    {
        $this->setCode(static::STATUS_CODE_EMAIL_NOT_PROVIDED);

        return $this;
    }
```

Figura 4 7.1.5.4.1 Classe InvalidCommandException estat inicial

Finalment es llencen tots els tests de la classe SignInCommandTest per comprovar que funcionen.

```
#!/usr/bin/env php
PHPUnit 5.0.10 by Sebastian Bergmann and contributors.

.                                                                   1 / 1 (100%)

Time: 186 ms, Memory: 3.25Mb

OK (1 test, 1 assertion)
```

Figura 5 7.1.5.4.1 Classe InvalidCommandException estat inicial

Una vegada comprovat que els tests previs funcionen, podem executar el següent pas el qual és escriure un nou test que falli per a poder escriure codi que defineixi una millora o una nova funcionalitat pel sistema.

VERMELL

Un cop verificat que estem en un estat inicial correcte, és a dir, on tots els tests existents funcionen, es pot escriure un nou test que no funcioni.

En l'apartat anterior s'ha pogut observar que el paràmetre *$email* de la funció *__construct* de la classe *SingInCommand* no podia ser una cadena de text buida. En aquest apartat **es vol afegir** a més la validació de què el camp *$email* tingui un **format vàlid d'adreça de correu electrònic**.

A continuació, es procedeix a escriure el test que expressi aquesta nova funcionalitat. El nom del test és *commandWithNotValidEmailShouldThrowExceptionWithEmailNotValidProvidedStatusCode*. Consisteix a invocar el mètode *buildCommand* del test el qual s'encarrega d'invocar el mètode *__construct* de la classe *SignInCommand*, amb un paràmetre d'entrada *$email* que no té un format correcte d'adreça electrònica. En aquest cas, el métode *__construct* de la classe *SignInCommand* hauria de llençar una excepció de tipus *InvalidCommandException* amb un codi d'excepció *STATUS_CODE_EMAIL_NOT_VALID_PROVIDED*.

```php
/**
 * @test
 */
public function commandWithNotValidEmailShouldThrowExceptionWithEmailNotValidProvidedStatusCode()
{
    try {
        $this->buildCommand('test');
        $this->thisTestFails();
    } catch (InvalidCommandException $invalidCommandException) {
        $this->assertEquals(
            InvalidCommandException::STATUS_CODE_EMAIL_NOT_VALID_PROVIDED,
            $invalidCommandException->code()
        );
    }
}
```

Figura 6 7.1.5.4.1 Nou test per comprovar si el format del email es vàlid

Un cop s'ha escrit el test, s'executa per veure que no funciona. En la imatge següent es pot observar que el test ni tan sols compila.

```
#!/usr/bin/env php
PHPUnit 5.0.10 by Sebastian Bergmann and contributors.

PHP Fatal error:  Undefined class constant 'STATUS_CODE_EMAIL_NOT_VALID_PROVIDED'
```

Figura 7 7.1.5.4.1 El nou test creat no funciona

En aquest punt és quan es realitza la següent fase, on el programador escriu el codi necessari per a fer que el test funcioni.

VERD

En aquesta fase s'ha d'escriure el mínim codi i de la forma més ràpida possible per fer que el test funcioni. És el concepte de baby step que s'ha introduït en apartats anteriors i que també aplica a la posterior fase de refactor.

En primer lloc, creem la constant en la classe *InvalidCommandException* per a què el codi compili.

```
class InvalidCommandException extends \Exception
{
    const STATUS_CODE_EMAIL_NOT_PROVIDED       = -1;
    const STATUS_CODE_EMAIL_NOT_VALID_PROVIDED = -2;
```

Figura 8 7.1.5.4.1 Codi per a que el test compili

S'executa el test per a veure quin és l'estat del sistema. Com es pot observar en la imatge següent, el test compila però segueix fallant. Ens informa que es rep un codi d'excepció valor -1 en comptes de valor -2 que és el que s'espera. Si s'observa la imatge anterior, els valors corresponen a les constants *STATUS_CODE_EMAIL_NOT_PROVIDED* i *STATUS_CODE_EMAIL_NOT_PROVIDED* respectivament de la classe *InvalidCommandException*. Aquest fet significa que hem de seguir afegint codi al sistema per a què el test funcioni.

```
PHPUnit 5.0.10 by Sebastian Bergmann and contributors.

F

Time: 191 ms, Memory: 3.50Mb

There was 1 failure:

1) Test\Interactor\CommandHandler\SignIn\SignInCommandTest
Failed asserting that -1 matches expected -2.
```

Figura 9 7.1.5.4.1 El test no funciona. Els valors rebuts i els esperats no coincideixen

Continuant amb el procés, s'opta per modificar la classe *SignInCommand* i en concret el mètode *__construct*. S'afegeix una condició invocant el mètode *filter_var* natiu del llenguatge PHP amb els paràmetres *$email* que es rep com a paràmetre d'entrada del mètode *__construct* i la constant *FILTER_VALIDATE_EMAIL* també nativa del llenguatge de programació. La resposta del mètode ens indicarà si el email és vàlid (retornarà el valor true) o pel contrari no ho és (retornarà el valor false). La condició la neguem per a què s'executi el flux de codi en cas que no sigui el format vàlid.

Aquest flux de codi consisteix a crear una nova excepció de tipus *InvalidCommandException* amb el paràmetre d'entrada el codi d'excepció *STATUS_CODE_EMAIL_NOT_VALID_PROVIDED* creat anterirorment.

```
public function __construct($email)
{
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
        throw new InvalidCommandException(
            InvalidCommandException::STATUS_CODE_EMAIL_NOT_VALID_PROVIDED
        );
    }
    throw new InvalidCommandException(
        InvalidCommandException::STATUS_CODE_EMAIL_NOT_PROVIDED
    );
}
```

Figura 10 7.1.5.4.1 Codi que valida si el format email és correcte

Executem el test per a comprovar en quin estat es troba el nostre sistema. Es pot observar en la imatge següent que el test segueix sense funcionar però, amb la diferència que el valor que es retorna en aquest cas és el 0 en comparació amb l'execució anterior la qual retornava el valor -1.

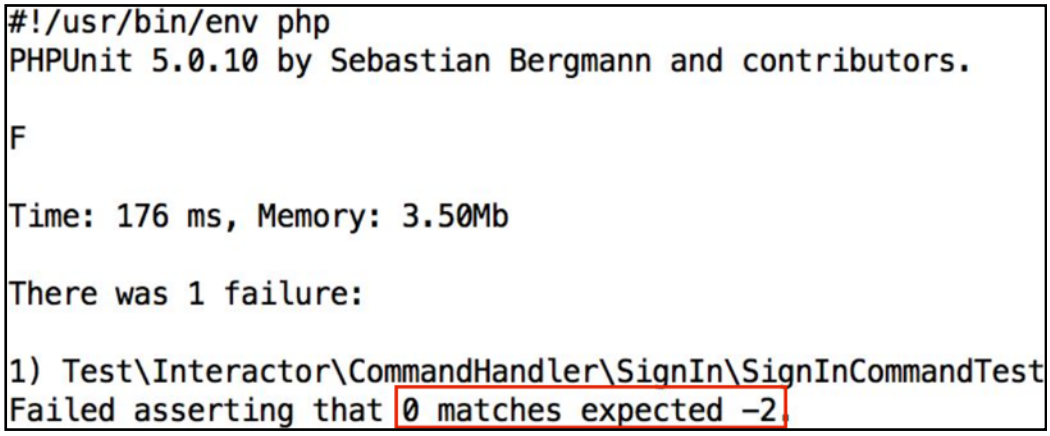


Figura 11 7.1.5.4.1 El test no funciona Els valors rebuts i els esperats no coincideixen

A continuació, s'opta per a modificar la classe *InvalidCommandException* afegint en el mètode *__construct* un cas més a la instrucció switch per a tractar el paràmetre *$statusCode* quan tingui el valor de la constant *STATUS_CODE_EMAIL_NOT_VALID_PROVIDED*. En aquest cas, s'invocarà al mètode privat *emailNotValidProvided* el qual també s'implementa en aquesta fase del procés. Aquest mètode a la vegada invoca el mètode privat de la mateixa classe *setCode* amb el paràmetre d'entrada la constant *STATUS_CODE_EMAIL_NOT_VALID_PROVIDED*. Per últim, el mètode *setCode*, modificarà el camp *code*. D'aquesta forma quan s'invoqui el mètode públic *code* de la mateixa classe, retornarà el valor del camp que anteriorment se li ha assignat.

```php
public function __construct($statusCode)
{
    switch ($statusCode) {
        case static::STATUS_CODE_EMAIL_NOT_PROVIDED:
            $this->emailNotProvided();
            break;
        case static::STATUS_CODE_EMAIL_NOT_VALID_PROVIDED:
            $this->emailNotValidProvided();
            break;
    }

    parent::__construct($this->message(), $this->code());
}

/**
 * @return InvalidCommandException
 */
private function emailNotProvided()
{
    $this->setCode(static::STATUS_CODE_EMAIL_NOT_PROVIDED);

    return $this;
}

/**
 * @return InvalidCommandException
 */
private function emailNotValidProvided()
{
    $this->setCode(static::STATUS_CODE_EMAIL_NOT_VALID_PROVIDED);

    return $this;
}
```

Figura 12 7.1.5.4.1 Codi del mètode emailNotValidProvided

A continuació, s'executa el test i el test funciona amb el codi creat fins ara.

```
#!/usr/bin/env php
PHPUnit 5.0.10 by Sebastian Bergmann and contributors.

.                                                                   1 / 1 (100%)

Time: 174 ms, Memory: 3.25Mb

OK (1 test, 1 assertion)
```

Figura 13 7.1.5.4.1 Execució segon test funciona

En aquest punt tenim dos testos creats. S'ha d'assegurar que tots els testos segueixen funcionant. S'executen els tests i s'observa que el primer test creat, el *commandWithoutEmailShouldThrowExceptionWithEmailNotProvidedStatusCode*, no funciona. Analitzant el perquè es pot notar que el valor del codi d'excepció que s'obté és el -2 en comptes de l'esperat que és el -1. És a dir, s'obté el valor de la constant *STATUS_CODE_EMAIL_NOT_VALID_PROVIDED* en comptes del valor esperat la constant *STATUS_CODE_EMAIL_NOT_PROVIDED*.

```
#!/usr/bin/env php
PHPUnit 5.0.10 by Sebastian Bergmann and contributors.

F.

Time: 172 ms, Memory: 3.50Mb

There was 1 failure:

1) Test\Interactor\CommandHandler\SignIn\SignInCommandTest
Failed asserting that -2 matches expected -1.
```

Figura 14 7.1.5.4.1 El test inicial no funciona

S'opta per modificar el codi de la classe *SignInCommand* i en concret el mètode *__construct* afegint una primera condició on es comprova si el paràmetre d'entrada *$email* està buit. En aquest cas, es crea una excepció de tipus I*nvalidCommandException* com a paràmetre d'entrada la constant *STATUS_CODE_EMAIL_NOT_PROVIDED*.

```
public function __construct($email)
{
    if (null === $email) {
        throw new InvalidCommandException(
            InvalidCommandException::STATUS_CODE_EMAIL_NOT_PROVIDED
        );
    }
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
        throw new InvalidCommandException(
            InvalidCommandException::STATUS_CODE_EMAIL_NOT_VALID_PROVIDED
        );
    }
}
```

Figura 15 7.1.5.4.1 Comprovar si el paràmetre $email és una cadena buida

Finalment, es torna a executar tots els testos i aquesta vegada es pot observar que tots ells funcionen, assolint, d'aquesta forma, l'objectiu de la fase actual.

```
#!/usr/bin/env php
PHPUnit 5.0.10 by Sebastian Bergmann and contributors.

..                                                                  2 / 2 (100%)

Time: 166 ms, Memory: 3.00Mb

OK (2 tests, 2 assertions)
```

Figura 16 7.1.5.4.1 Tots els test creats funcionen

En aquest punt del procés, és refactoritza el codi acabat que s'acaba d'implementar.

REFACTORITZAR

A continuació, s'analitza el codi de la classe *SignInCommand* que es pot refactoritzar. En primer lloc i per donar més valor semàntic al codi, es pren la decisió d'extreure tot el codi del mètode *__construct* a un mètode privat de la classe que té per nom *validateEmailParam*. D'aquesta forma se li atorga intencionalitat al codi fent-lo més fàcil d'entendre.

```php
use Interactor\CommandHandler\SignIn\Exception\InvalidCommandException;

class SignInCommand
{
    /**
     * @param string $email
     *
     * @throws InvalidCommandException
     */
    public function __construct($email)
    {
        $this->validateEmailParam($email);
    }

    /**
     * @param $email
     *
     * @throws InvalidCommandException
     */
    private function validateEmailParam($email)
    {
        if (null === $email) {
            throw new InvalidCommandException(
                InvalidCommandException::STATUS_CODE_EMAIL_NOT_PROVIDED
            );
        }
        if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
            throw new InvalidCommandException(
                InvalidCommandException::STATUS_CODE_EMAIL_NOT_VALID_PROVIDED
            );
        }
    }
}
```

Figura 17 7.1.5.4.1 Es crea el mètode validateParam de la classe SignInCommand

Per a cada pas que es realitza de refactorització de codi, és recomanable executar els tests, per a garantir que aquests segueixen funcionant i que el sistema es troba en un estat estable. D'aquesta forma, s'executen els tests i s'observa que aquests segueixen funcionant.

```
#!/usr/bin/env php
PHPUnit 5.0.10 by Sebastian Bergmann and contributors.

..                                                                  2 / 2 (100%)

Time: 166 ms, Memory: 3.00Mb

OK (2 tests, 2 assertions)
```

Figura 18 7.1.5.4.1 Execució tests després del refactor de codi

La següent decisió que s'ha pres ha estat la d'extreure el codi de la condició *!filter_var($email, FILTER_VALIDATE_EMAIL)* en un mètode que té el nom de *notValidEmail*. D'aquesta forma i com en el primer refactor que s'ha realitzat, se li atorga al codi major intencionalitat i contingut semàntic fent-lo, d'aquesta forma, més intel·ligible.

```php
/**
 * @param $email
 *
 * @throws InvalidCommandException
 */
private function validateEmailParam($email)
{
    if (null === $email) {
        throw new InvalidCommandException(
            InvalidCommandException::STATUS_CODE_EMAIL_NOT_PROVIDED
        );
    }
    if ($this->notValidEmail($email)) {
        throw new InvalidCommandException(
            InvalidCommandException::STATUS_CODE_EMAIL_NOT_VALID_PROVIDED
        );
    }
}

/**
 * @param $email
 *
 * @return bool
 */
private function notValidEmail($email)
{
    return !filter_var($email, FILTER_VALIDATE_EMAIL);
}
```

Figura 19 7.1.5.4.1 Es crea el mètode privat notValidEmail de la clase SignInCommand

Com en el pas anterior, es tornen a executar els tests per comprovar que funcionen i efectivament segueixen funcionant. D'aquesta forma es pot prosseguir a continuar el procés amb garanties de què el sistema es troba en un estat estable.

```
#!/usr/bin/env php
PHPUnit 5.0.10 by Sebastian Bergmann and contributors.

..                                                                  2 / 2 (100%)

Time: 166 ms, Memory: 3.00Mb

OK (2 tests, 2 assertions)
```

Figura 20 7.1.5.4.1 Execució tests després del refactor de codi

A continuació, es pren la decisió d'eliminar el codi duplicat. Es pot observar que la creació de l'excepció es repeteix el nom de la classe *InvalidCommandException*, D'aquesta forma, es decideix extreure la creació de la classe en un mètode privat que rep com a paràmetre d'entrada el codi d'excepció i el qual té el nom de *buildInvalidCommandException*.

```
private function validateEmailParam($email)
{
    if (null === $email) {
        throw $this->buildInvalidCommandException(
            InvalidCommandException::STATUS_CODE_EMAIL_NOT_PROVIDED
        );
    }
    if ($this->notValidEmail($email)) {
        throw $this->buildInvalidCommandException(
            InvalidCommandException::STATUS_CODE_EMAIL_NOT_PROVIDED
        );
    }
}

/**
 * @param string $email
 *
 * @return bool
 */
private function notValidEmail($email)
{
    return !filter_var($email, FILTER_VALIDATE_EMAIL);
}

/**
 * @param int $statusCode
 *
 * @return InvalidCommandException
 */
private function buildInvalidCommandException($statusCode)
{
    return new InvalidCommandException(
        $statusCode
    );
}
```

Figura 21 7.1.5.4.1 Creació del mètode buildInvalidCommandException

A continuació s'executen els tests per a garantir que el sistema segueix tenint un estat estable, tot i els canvis realitzats en el codi. Els tests segueixen funcionant.

```
#!/usr/bin/env php
PHPUnit 5.0.10 by Sebastian Bergmann and contributors.

..                                                                  2 / 2 (100%)

Time: 166 ms, Memory: 3.00Mb

OK (2 tests, 2 assertions)
```

Figura 22 7.1.5.4.1 Execució tests després del refactor de codi

Finalment és **refactoritza el codi dels tests**. És molt important que també els tests estiguin implementats amb codi net i intel·ligible així com evitar la duplicació de codi per a facilitar el seu manteniment.

Es pot observar que en el test *commandWithNotValidEmailShouldThrowExceptionWithEmailNotValidProvidedStatusCode* existeix una string 'test'. S'extreu aquest string en una constant amb nom *EMAIL_NOT_VALID_TEST*. D'aquesta forma, com en refactors anteriors, dotem al codi d'intencionalitat i de valor semàntic per a facilitar la seva comprensió.

```php
const EMAIL_NOT_VALID_TEST = 'test';

/**
 * @test
 */
public function commandWithNotValidEmailShouldThrowExceptionWithEmailNotValidProvidedStatusCode()
{
    try {
        $this->buildCommand(self::EMAIL_NOT_VALID_TEST);
        $this->thisTestFails();
    } catch (InvalidCommandException $invalidCommandException) {
        $this->assertEquals(
            InvalidCommandException::STATUS_CODE_EMAIL_NOT_VALID_PROVIDED,
            $invalidCommandException->code()
        );
    }
}
```

Figura 23 7.1.5.4.1 Refactor codi del tests

Finalment, com a cada refactor, es comprova que els tests segueixen funcionant.

```
#!/usr/bin/env php
PHPUnit 5.0.10 by Sebastian Bergmann and contributors.

..                                                                  2 / 2 (100%)

Time: 166 ms, Memory: 3.00Mb

OK (2 tests, 2 assertions)
```

Figura 24 7.1.5.4.1 Execució tests després del refactor de codi

7.1.6. Infrastructure

Des de l'inici de la descripció de l'arquitectura del software fins a aquest apartat s'ha de destacar un detall molt important: **en cap moment s'ha parlat de la tecnologia escollida per implementar les funcionalitats del sistema**.

La capa Infrastructure és on es defineixen i s'implementen els **components software els quals estan lligats directament amb la tecnologia escollida**. Basicament, es tracten dels adapters secundaris que implementen les interfícies del adapters primaris de la capa Domain. Aquests components interactuen exclusivament amb els CommandHandlers de la capa Interactor, els quals invoquen els mètodes definits en els Repositoris i DataTransformers de la capa Domain, els quals estan implementats pels Repositoris i DataTransformers de la capa Infrastructure.

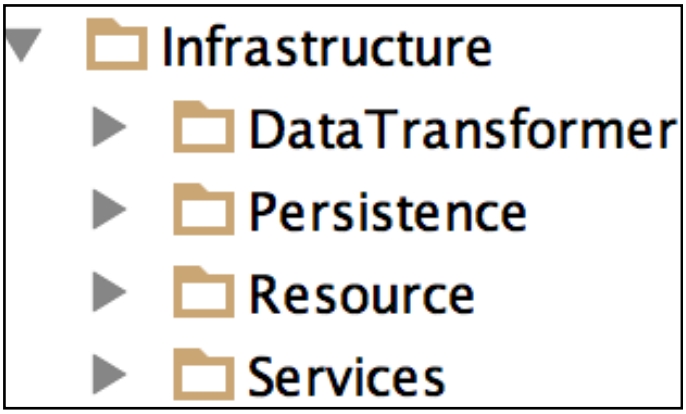


Figura 1 7.1.6 Infrastructure directoris

7.1.6.1. Repositoris

Són els components software que **s'encarreguen de gestionar les entitats de la capa de Domini amb la tecnologia escollida per a realitzar la seva persistència i consulta**. Implementen les interfícies dels repositoris de la capa Domain. Els seus **mètodes d'accés públic han de coincidir amb els mètodes de les interfícies** per a mantenir el contracte entre les dues capes.

S'ha establert la convenció de què els noms dels repositoris de la capa Infrastructure tinguin com a prefix el nom de la tecnologia la qual els implementa i la resta sigui el nom de la interfície de la capa Domain. Per exemple, el BadgeRepository de la capa Domain tindrà el nom TecnologyBadgeRepository on Tecnology és la tecnologia escollida per a implementar el repositori.

Es troben ubicats dins el directori Infrastructure en el directori Persistence. Cada directori té el nom de la tecnologia emprada per implementar el repositori en qüestió.

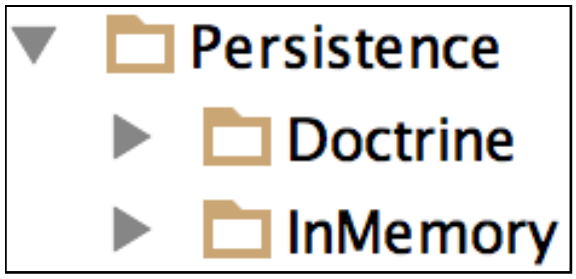


Figura 2 7.1.6 Infrastructure repositoris

7.1.6.1.1. InMemory

Aquests repositoris s'implementen utilitzant la memòria RAM del PC mitjançant les estructures natives del llenguatge de programació taules o arrays. Aquestes són estructures que representen una col·lecció d'objectes en memòria.

Dins del directori InMemory es replica l'estructura de directoris de la capa Domain per ubicar els repositoris. És a dir, existeix la ruta Domain/Entity i dins una carpeta per a cada entitat on s'ubica el repositori associat.

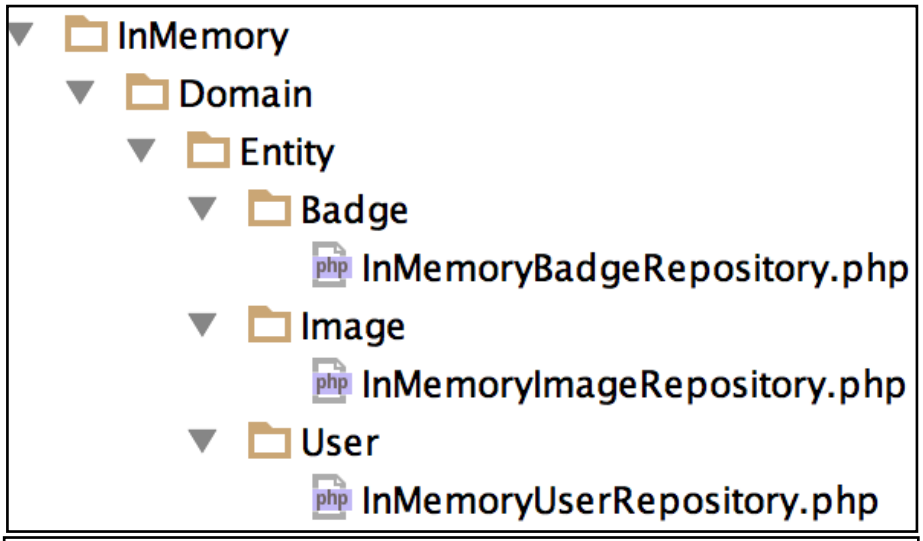


Figura 1 7.1.6.1.1 InMemory directoris

S'il·lustra la implementació amb l'exemple del *InMemoryImageRepository*.

```php
interface ImageRepository
{
    /**
     * @param Image $image
     */
    public function persist(Image $image);

    /**
     * @param string $id
     *
     * @return Image | null
     */
    public function find($id);

    /**
     * @param Image $image
     */
    public function remove(Image $image);
}
```

Figura 2 7.1.6.1.1 ImageRepository capa Domain

```php
class InMemoryImageRepository implements ImageRepository
{
    /**
     * @var Image[]
     */
    private $images;

    public function __construct($images = [])
    {
        $this->images = $images;
    }

    public function persist(Image $image)
    {
        $this->images[] = $image;
    }

    /**
     * {@inheritdoc}
     */
    public function find($id)
    {
        $aNullImage = null;
        foreach ($this->images as $image) {
            if ($image->id() === $id) {
                return $image;
            }
        }

        return $aNullImage;
    }

    /**
     * {@inheritdoc}
     */
    public function remove(Image $image)
    {
        $aNullImage = null;
        foreach ($this->images as $imageIndex => $anImage) {
            if ($image->id() === $anImage->id()) {
                unset($this->images[$imageIndex]);
                return;
            }
        }
    }
}
```

Figura 3 7.1.6.1.1 InMemoryImageRepository capa Infrastructure

Com es pot observar, la classe **InMemoryImageRepository implementa la interfície ImageRepository**. Comparteixen els mateixos mètodes públics. InMemoryImageRepository en si és una col·lecció que conté entitats de domini de tipus Image.

Els repositoris implementats amb aquesta tecnologia, són utilitzats pels tests unitaris a manera de repositoris mockery. La **tècnica mockery s'utilitza per a què els tests unitaris no depenguin de la tecnologia**. D'aquesta forma, els tests unitaris tan sols testejen aquell codi al qual estan associats, no la infraestructura depenent de la tecnologia escollida. Aquests **són injectats per dependència al mètode constructor dels CommandHandlers**. S'il·lustra aquest exemple amb un test creat en la classe CreateBadgeCommandHandlerTest.

```php
public function exceptionRepositoryWhenFindUserShouldThrownExceptionBadgeNotCreatedStatusCode()
{
    try {
        $userRepository = $this->buildFakeUserRepositoryThrownException(
            $this->buildDefaultUsers(),
            FakeUserRepositoryThrownException::FIND_BY_ID_METHOD_THROW_EXCEPTION
        );
        $commandHandler = $this->buildCreateBadgeCommandHandler(
            $userRepository,
            $this->buildImageRepository(),
            $this->buildBadgeRepository(),
            $this->buildIdGenerator(),
            $this->buildImageManager(),
            $this->buildBadgeDataTransformer()
        );
        $commandHandler->handle($this->buildCommand());

        $this->thisTestFails();
    } catch (InvalidCreateBadgeCommandHandlerException $invalidCreateBadgeCommandHandlerException) {
        $this->assertEquals(
            InvalidCreateBadgeCommandHandlerExceptionCode::STATUS_CODE_BADGE_NOT_CREATED,
            $invalidCreateBadgeCommandHandlerException->code()
        );
    }
}
```

Figura 4 7.1.6.1.1 CreateBadgeCommandHandlerTest creació command handler i injecció ImageRepository

```php
private function buildCreateBadgeCommandHandler(
    UserRepository       $userRepository,
    ImageRepository      $imageRepository,
    BadgeRepository      $badgeRepository,
    IdGenerator          $idGenerator,
    ImageManager         $imageManager,
    BadgeDataTransformer $badgeDataTransformer
) {
    return new CreateBadgeCommandHandler(
        $userRepository,
        $imageRepository,
        $badgeRepository,
        $idGenerator,
        $imageManager,
        $badgeDataTransformer
    );
}
```

Figura 5 7.1.6.1.1 Mètode buildCreateBadgeCommandHandler

```php
/**
 * @return InMemoryImageRepository
 */
private function buildImageRepository()
{
    return new InMemoryImageRepository();
}
```

Figura 6 7.1.6.1.1 Mètode buildImageRepository

S'ha de notar que en el constructor del CreateBadgeCommandHandler se l'informa d'un objecte de tipus ImageRepository, no d'un objecte InMemoryRepository. D'aquesta forma es manté l'abstracció i desacoblament amb la tecnologia. Qualsevol repositori de la capa Infrastructure que implementi la interfície ImageRepository es pot injectar en aquest CommandHandler. A continuació es pot observar la invocació per part del CreateBadgeCommandHandler d'un mètode del ImageRepository el qual implementa en aquest cas el InMemoryImageRepository.

```php
private function tryToPersistImageData(Image $image)
{
    try {
        $this->imageRepository->persist($image);
    } catch (\Exception $exception) {
        throw $this->buildInvalidCreateBadgeCommandHandlerException(
            InvalidCreateBadgeCommandHandlerExceptionCode::STATUS_CODE_BADGE_NOT_CREATED
        );
    }
}
```

Figura 7 7.1.6.1.1 Invocació mètode persist del ImageRepository en el CreateBadgeCommandHandler

7.1.6.1.2. Mysql Doctrine ORM

La tecnologia escollida per a realitzar la persistència de dades del sistema es MySQL [44]. Es tracta d'un sistema de gestió de bases de dades relacional desenvolupada actualment per Oracle Corporation i es tracta de la base de dades open source més estesa del món. Té una gran comunitat que dòna suport i la utilitzen empreses importants com Facebook i Twitter. En el projecte no s'explotarà totes les funcionalitats que ofereix però l'elecció també es realitza considerant l'escalabilitat del sistema.

S'utilitza el ORM (object relational mapping) Doctrine [45], el qual ens servirà com a capa d'abstracció entre el codi i la base de dades. Doctrine permet mitjançant fitxers de configuració relacionar els camps i les taules de la base de dades amb les entitats definides del codi software. Un cop realitzada aquesta configuració, mitjançant l'entity manager, el qual conté la configuració per connectar amb la base de dades, es pot gestionar les operacions bàsiques com guardar o esborrar entitats sense necessitat d'escriure sentencies SQL.

S'il·lustra la configuració d'una entitat amb l'exemple de l'entitat de domini Image, el seu fitxer de configuració i la taula de la base de dades.

```
class Image
{
    /** @var  string */
    private $id;
    /** @var  string */
    private $name;
    /** @var  int */
    private $width;
    /** @var  int */
    private $height;
    /** @var  string */
    private $format;
```

Figura 1 7.1.6.1.2 Image Atributs

id	VARCHAR
name	VARCHAR
width	INT
height	INT
format	VARCHAR

Figura 2 7.1.6.1.2 Camps taula Image base de dades Mysql

A continuació es mostra la implementació del DoctrineImageRepository.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<doctrine-mapping xmlns="http://doctrine-project.org/schemas/orm/doctrine-mapping"
                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xsi:schemaLocation="http://doctrine-project.org/schemas/orm/doctrine-mapping
        http://doctrine-project.org/schemas/orm/doctrine-mapping.xsd">
    <entity name="Domain\Entity\Image\Image">
        <id name="id" type="string" length="200" />
        <field name="name" type="string" length="200" />
        <field name="width" type="integer" />
        <field name="height" type="integer" />
        <field name="format" type="string" length="5" />
    </entity>
</doctrine-mapping>
```

Figura 3 7.1.6.1.2 Doctrine Image configuració

```php
interface ImageRepository
{
    /**
     * @param Image $image
     */
    public function persist(Image $image)

    /**
     * @param string $id
     *
     * @return Image | null
     */
    public function find($id);

    /**
     * @param Image $image
     */
    public function remove(Image $image);
}
```

Figura 4 7.1.6.1.2 Domain ImageRepository

```php
class DoctrineImageRepository implements ImageRepository
{
    /** @var EntityManager */
    private $entityManager;

    /** @var \Doctrine\ORM\EntityRepository */
    private $entityRepository;

    public function __construct($entityManager)
    {
        $this->entityManager    = $entityManager;
        $this->entityRepository = $this->entityManager->getRepository(Image::class);
    }

    public function persist(Image $image)
    {
        $this->entityManager->persist($image);
        $this->entityManager->flush($image);
    }

    /**
     * @param string $id
     *
     * @return Image | null
     */
    public function find($id)
    {
        return $this->entityRepository->find($id);
    }

    /**
     * @param Image $image
     */
    public function remove(Image $image)
    {
        $this->entityManager->remove($image);
        $this->entityManager->flush($image);
    }
}
```

Figura 5 7.1.6.1.1 DoctrineImageRepository

Mitjançant la classe Entity Manager es pot utilitzar totes les funcionalitats que ofereix Doctrine. Per exemple, s'ha de notar que el mètode find el qual donat un $id de l'entitat la cerca a la base de dades, s'implementa invocant al mètode find del Entity Repository. Mitjançant l'abstracció, aquest component sap com connectar-se a la base de dades, quina taula conté la informació i construeix una query SQL de tipus Select per obtenir la informació necessària. Un cop té la resposta de la base de dades, crea una nova entitat de tipus Image mapejant la informació dels camps retornats per la base de dades amb els atributs de l'entitat.

Actualment, existeixen **tres taules a les bases de dades que representen les entitats Badge, User i Image** respectivament i tenen els mateixos camps que els atributs de les entitats i el nom de la taula és el mateix que el de l'entitat. Les associacions, per exemple, entre les entitats Badge i User es realitza mitjançant clau forana en la taula Badge cap a la taula User. També hi ha una clau forana en la taula Badge que relaciona aquesta taula amb la taula Image.

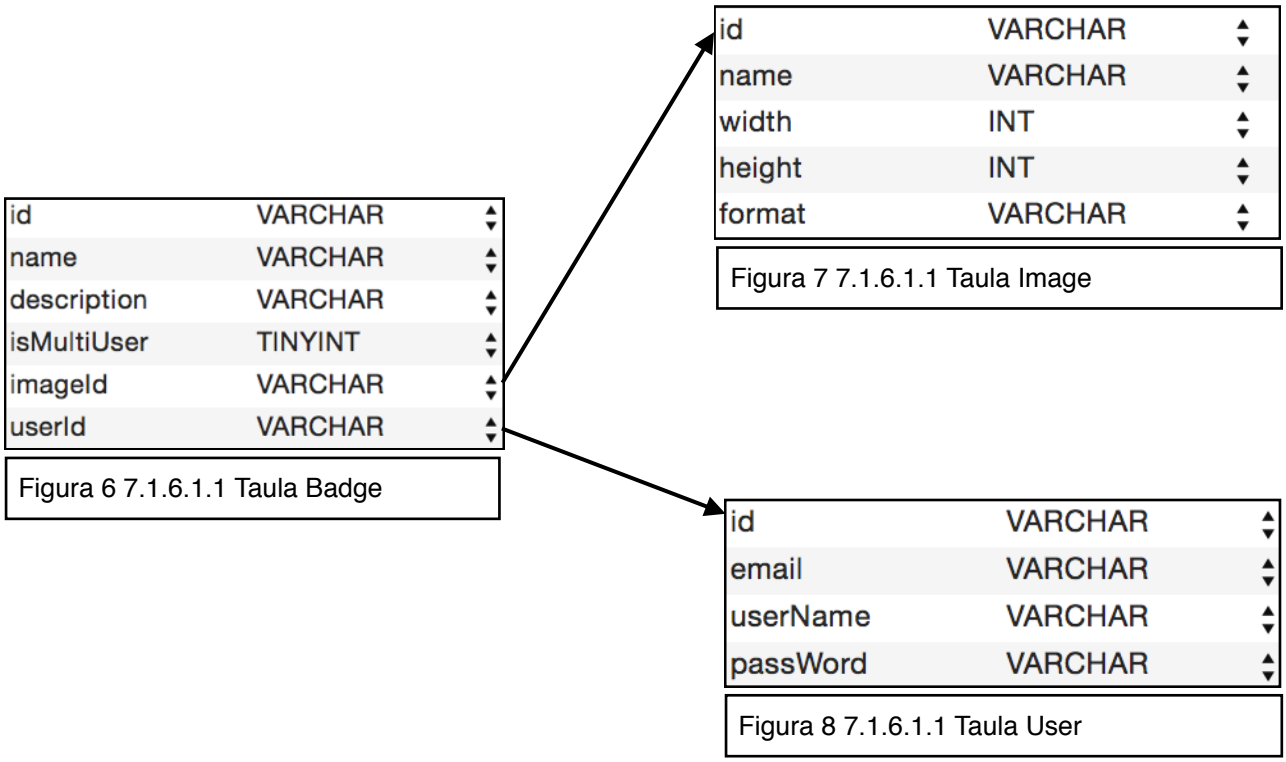


Figura 6 7.1.6.1.1 Taula Badge

Figura 7 7.1.6.1.1 Taula Image

Figura 8 7.1.6.1.1 Taula User

Els repositoris es troben dins del directori Doctrine/Domain/Entity i el nom de l'entitat al que estan associats. Els fitxers de configuració per relacionar les entitats amb les seves respectives taules es troben ubicats en el directori Doctrine/Mapping i tenen el nom per convenció de NomEntitat.NomEntitat.orm.xml. Aquesta és la tecnologia escollida i la qual serà utilitzada per persistir les dades per part del sistema.

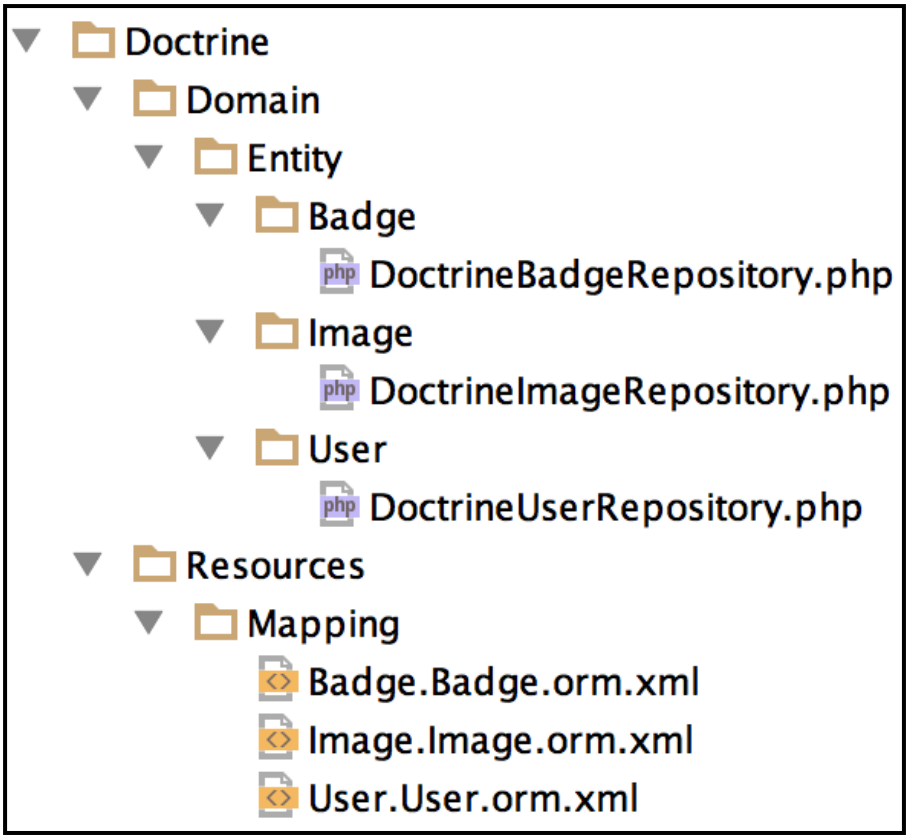


Figura 7 7.1.6.1.1 Ubicació DoctrineRepositoris i el mappeig d'entitats

7.1.6.2. Data Transformers

Aquests components seran els encarregats de **convertir una entitat de domini en un objecte de tipus resource** per a poder exposar la informació que conté l'entitat de domini sense exposar la mateixa entitat. Implementen les interfícies *DataTransformers* de la capa de domini els quals tan sols tenen un mètode públic que es diu *transform,* el qual rep com a paràmetre d'entrada l'entitat o col·lecció d'entitats que es volen transformar en *resources*.

Es troben ubicats dins el directori *DataTransformer*. En aquest directori hi ha els de tipus *NoOperation* i els de tipus *Resource*.

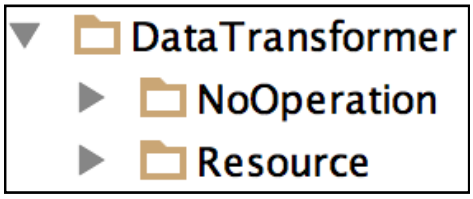


Figura 1 7.1.6.2 Directoris

7.1.6.2.1. NoOperation

Els DataTransformers de tipus *NoOperation* s'utilitzen a monera de mockery per implementar unit tests. En l'àmbit de test es pren la decisió de què els *DataTransformers* retornin la mateixa entitat per no dependre dels resources i facilitar la implementació dels tests fent-los independents de la infraestructura. Són utilitzats en els units tests dels *CommandHandlers*. Estan ubicats dins el directori *NoOperation* en la ruta Domain/Entity amb el nom de l'entitat associada.

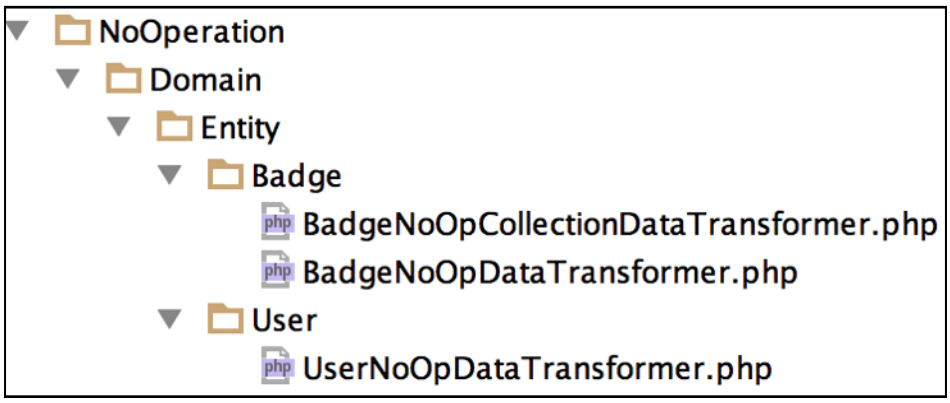


Figura 1 7.1.6.2.1 NoOperation directoris

A continuació, es mostra, a manera d'exemple, la implementació del *UserNoOpDataTransformer*.

```
interface UserDataTransformer
{
    /**
     * @param User $user
     *
     * @return mixed
     */
    public function transform(User $user);
}
```

Figura 2 7.1.6.2.1 UserDataTransformer interface capa Domain

```php
class UserNoOpDataTransformer implements UserDataTransformer
{
    /**
     * {@inheritdoc}
     */
    public function transform(User $user)
    {
        return $user;
    }
}
```

Figura 3 7.1.6.2.1 UserNoOpDataTransformer infrastructure implementació

A continuació, s'il·lustra amb un test de la classe *SignUpCommandHandlerTest* com s'injecta el *UserNoOpDataTransformer* a la classe *SignUpCommandHandler* i com es fa servir dins el *CommandHandler*.

```php
public function emailAlreadyExistsShouldThrowExceptionEmailAlreadyExistsStatusCode()
{
    try {
        $commandHandler = $this->buildCommandHandler($this->buildDefaultUserRepository());
        $commandHandler->handle(
            $this->buildCommand(
                static::EMAIL_EXISTS_TEST_BADGES_IO_COM,
                static::USERNAME_EXISTS_VALID_BADGES_USER,
                static::PASSWORD_VALID_BE_FREE
            )
        );
        $this->thisTestFails();
    } catch (InvalidSignUpCommandHandlerException $invalidSignUpCommandHandlerException) {
        $this->assertEquals(
            InvalidSignUpCommandHandlerExceptionCode::STATUS_CODE_EMAIL_ALREADY_EXISTS,
            $invalidSignUpCommandHandlerException->code()
        );
    }
}
```

Figura 4 7.1.6.2.1 Mètode buildCommandHandler de la classe SignUpCommandHandlerTest

```
private function buildCommandHandler(UserRepository $userRepository)
{
    return new SignUpCommandHandler(
        $userRepository,
        $this->buildUserDataTransformer(),
        $this->buildIdGenerator(),
        $this->buildPasswordCipher()
    );
}
```

Figura 5 7.1.6.2.1 Mètode buildUserDataTransformer de la classe SingUpCommadnHandlerTest

```
private function buildUserDataTransformer()
{
    return new UserNoOpDataTransformer();
}
```

Figura 6 7.1.6.2.1 Creació UserNoOpDataTransformer

```
public function handle($command)
{
    $this->validate($command);
    $user = $this->buildUser($command);
    $this->tryToPersistUser($user);

    return $this->userDataTransformer->transform($user);
}
```

Figura 7 7.1.6.2.1 Incovació mètode transform UserDataTransformer

7.1.6.2.2. Resource

Aquest tipus d'implementació de *DataTransformer* converteix una entitat en un objecte de tipus Resource. El *Resource* és un objecte que conté la informació necessària per part de l'entitat. Aquesta informació serà exposada a capes més externes. En el context del projecte, els components de la capa App seran els que gestionaran aquesta informació, la processaran i la mostraran a l'usuari.

Els DataTransformers Es troben ubicats en el directori Resource/Domain/Entity i el nom de l'entitat associada. Pel que fa als objecteds resource és toven en el directori Infraestructure/Resource/Domain/

Entity i el nom de l'entitat al que estan associats. En el context del projecte **és la tecnologia escollida per a comunicar la informació de les entitats de la capa Interactor a la capa App sense exposar els objectes del nucli del sistema**.

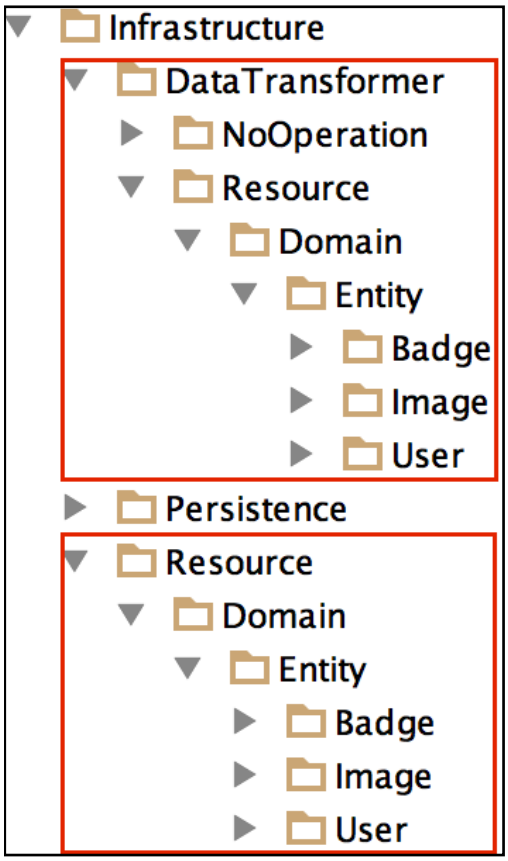


Figura 1 7.1.6.2.2 Resource directoris

```php
interface UserDataTransformer
{
    /**
     * @param User $user
     *
     * @return mixed
     */
    public function transform(User $user);
}
```

Figura 2 7.1.6.2.2 UserDataTransformer interficie capa Domain

A continuació, es mostra la implementació del *UserResourceDataTransformer* així com la del *UserResource*.

```php
class UserResourceDataTransformer implements UserDataTransformer
{
    /**
     * {@inheritdoc}
     */
    public function transform(User $user)
    {
        return new UserResource(
            $user->id(),
            $user->email(),
            $user->userName(),
            $user->passWord()
        );
    }
}
```

Figura 3 7.1.6.2.2 UserResourceDataTransformer implementació capa Infrastructure

Es pot observar que *UserResourceDataTransformer* crea un objecte de tipus *UserResource* passant-li com a paràmetres d'entrada al mètode constructor la informació de l'entitat *User,* el qual rep per paràmetre el mètode *transform*. L'objecte UserResource té com atributs id, email, userName i passWord. Aquest tipus d'objecte no tenen validacions ni lògica de negoci associada. Aquest tipus d'objectes que tan sols contenen informació, en terminologia del software se solen anomenar com *data transfer objects (DTO)* [46].

```php
class UserResource
{
    /** @var  string */
    private $id;
    /** @var  string */
    private $email;
    /** @var  string */
    private $username;
    /** @var  string */
    private $password;

    public function __construct($id, $email, $username, $password)
    {
        $this->setId($id)
             ->setEmail($email)
             ->setUsername($username)
             ->setPassword($password);
    }
```

Figura 4 7.1.6.2.2 UserResource capa Infraestructure

7.1.6.3. Services

Finalment, els serveis d'infraestructura implementen els **serveis de la capa de domini**. S'ubiquen dins el directori Services/Domain i el directori amb el nom de cada servei. Actualment n'hi ha 3 implementacions, una per a cada servei de domini. Aquestes implementacions són:

- *UuIdGenerator*
  Servei de generadors de id el qual utilitza la llibreria ramsey/uuid de php
- *DiskImageManager*
  Servei que gestiona les imatges físiques associades als Badges les quals es persisteixen en el disc dur.
- *MD5PasswordCipher*
  Servei que encripta passwords mitjançant la llibreria MD5.

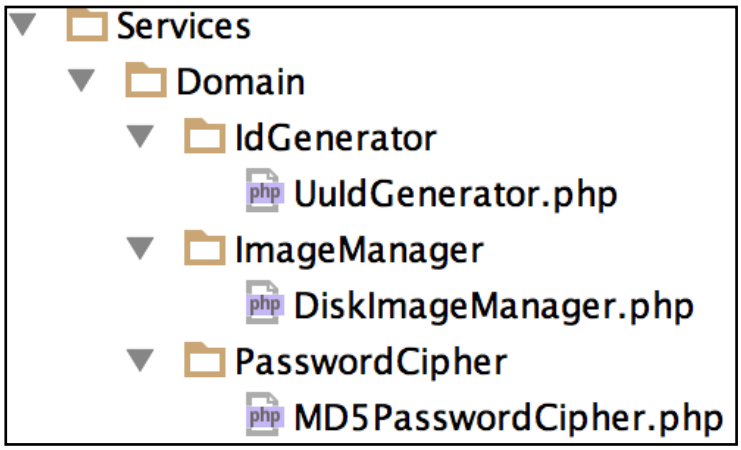


Figura 1 7.1.6.3 Services capa Infraestructure

S'il·lustra a manera d'exemple la implementació del servei MD5PasswordCipher.

```php
interface PasswordCipher
{
    /**
     * @param string $password
     *
     * @return string
     */
    public function cipher($password);
}
```

Figura 2 7.1.6.3 PasswordCipher capa Domain

```php
class MD5PasswordCipher implements PasswordCipher
{
    /**
     * {@inheritdoc}
     */
    public function cipher($password)
    {
        return md5($password);
    }
}
```

Figura 3 7.1.6.3 MD5PasswordCipher capa Infraestructure

7.1.7. App

La capa **App** conté els **components software els quals interaccionen amb els usuaris**. **Sota la demanda dels usuaris, aquests components interaccionen amb la capa Interactor invocant els mètodes *handle* dels *CommandHandlers*, passant-li com a paràmetre les dades transmeses pels usuaris i retornant el resultat de l'execució**.

Són els encarregats de realitzar la injecció de dependències als *CommandHandlers,* escollint la tecnologia adequada de la capa *Infrastructure* dels *Repositoris* i *DataTransformers* per a realitzar l'execució així com construir el *Command* amb tota la informació necessària per invocar els mètodes *handle*. **Però en cap concepte, els components de la capa App poden invocar mètodes dels components de la capa Infrastructure**. Aquests mètodes s'invocaran des dels *CommandHandlers* de la capa *Interactor*, com s'ha descrit en l'apartat on s'ha descrit la capa *Interactor*.

Finalment, s'encarregaran de la **presentació, oferint un *GUI* (Graphic User Interface) per a què l'usuari pugui interactuar amb el sistema i per mostrar els resultats generats per aquesta interacció**. S'ha de tenir en compte que una *GUI* potser tant un terminal de consola, com una pàgina web com un end point d'una API Rest. En el context del projecte, un usuari potser tant de naturalesa humana com un sistema automatitzat.

Identificant tipus de components de l'arquitectura hexagonal, els components software de la capa *App* (**ports secundaris**) invoquen els mètodes *handle* (**ports primaris**) dels *CommandHandlers*. Per a crear els *CommandHandlers*, s'escull quina tecnologia és la més adequada per a cada context per als *Repositoris* i *DataTransformers* (**adapters secundaris**) per injectar-los com a dependències i poder executar, finalment, la funcionalitat demandada.

La tecnologia escollida per servir el contingut generat per la capa App és el **servidor web Nginx** [47]. Es tracta d'un servidor web lleuger d'alt rendiment. Està implementat amb software lliure i és de codi obert. És multiplataforma, pel que pot executar-se en entorns tant de tipus Unix com de tipus Windows. L'utilitza empreses tan populars com poden ser WordPress, Netflix, Github o Facebook.

7.1.7.1. Symfony Framework

S'ha escollit el **framework Symfony** [48] per a implementar els components software de la capa App. Aquest framework, lliberat el 2005 pel seu creador Fabien Potencier, permet dissenyar aplicacions web. En un principi, estava dissenyat per crear aplicacions amb el patró MVC (model-vista-controlador) però gràcies a la seva gran flexibilitat es pot utilitzar en qualsevol tipus d'arquitectura. Proporciona moltes eines i classes encaminades a reduir el temps de desenvolupament d'una aplicació web complexa. **L'objectiu principal del framework és el de separar la lògica de negoci, de la lògica de servidor i de la presentació, desacoblant els tres contexts del sistema**.

Un dels conceptes clau d'aquest framework és la definició de **Bundle** [49]. Un Bundle és un conjunt estructurat d'arxius els quals s'ubiquen en un directori i implementen un tipus de funcionalitat determinada.

Els Bundles es poden empaquetar i ser distribuïts com a llibreries software. Aquest fet és molt interessant, ja que hi ha una gran comunitat que utilitza el framework per a poder crear llibreries com a Bundles els quals són distribuïts com a software de tercers i s'integren molt fàcilment amb altres aplicacions que utilitzen Symfony com a framework.

En el context del projecte, els components de la capa App estan agrupats per Bundles. Aquests estaran ubicats dins la carpeta App/Bundle.

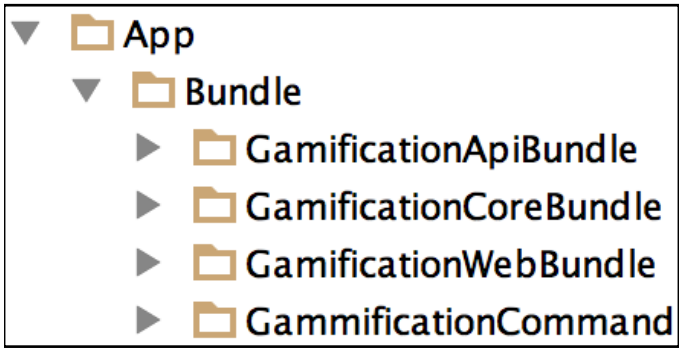


Figura 1 7.1.7.1 App Bundle directories

7.1.7.2. Injecció de dependències: Service Container

Un dels components més interessants que ofereix el framework Symfony és el ***Service Container o Dependency Injection Container*** [50].

Els components software de la capa App tenen la responsabilitat d'escollir quins Respositoris, DataTransformers i Serveis de la capa Infrastructure s'injecten als CommandHandlers. És a dir, **decidir quines implementacions tecnològiques són les més adequades en cada context per a poder realitzar una funció concreta**. S'ha pogut observar en la descripció de la capa Interactor que els **CommandHandlers reben per paràmetre en el mètode constructor les dependències necessàries** per a executar la lògica de negoci que tenen implementada. D'aquesta forma i a manera d'exemple, el *GetBadgeCommandHandler* se l'injecta com a dependències en el mètode constructor el *BadgeRepository* i el *BadgeDataTransformer*. S'ha de tenir en compte, que aquestes dependències poden tenir a la vegada altres dependències anidades,

complicant el procés d'instanciació d'objectes i dificultant la comprensió i reutilització del codi software.

```
public function __construct(
    BadgeRepository $badgeRepository,
    BadgeDataTransformer $badgeDataTransformer
) {
    $this->badgeRepository      = $badgeRepository;
    $this->badgeDataTransformer = $badgeDataTransformer;
}
```

Figura 1 7.1.7.2 Mètode constructor GetBadgeCommandHandler

S'ha de notar que el nom de les classes injectades són els de les interfícies (adpters primaris) per a mantenir el desacoblament i l'abstracció entre la capa Interactor i la capa Infrastructure.

Per a poder realitzar la injecció de dependències, els **components software de la capa App han d'instanciar el CommandHandler i han d'instanciar els components necessaris de la capa Infraestructure i injectar-los a través del constructor del CommandHanlder**. A continuació es mostra el codi el qual descriu aquest escenari amb l'exemple del *GetBadgeCommandHandler*.

```
$entityManager = new EntityManager();
$badgeRepository = new DoctrineBadgeRepository($entityManager);
$imageManager = new DiskImageManager();
$imageResourceDataTransformer = new ImageResourceDataTransformer(
    $imageManager
);
$userResourceDataTransformer = new UserResourceDataTransformer();
$badgeDataTransformer = new BadgeResourceDataTransformer(
    $imageResourceDataTransformer,
    $userResourceDataTransformer
);

$getBadgeCommandHandler = new GetBadgeCommandHandler(
    $badgeRepository,
    $badgeDataTransformer
);
```

Figura 2 7.1.7.2 Procés instanciació GetBadgeCommandHandler

Com es pot observar, el component D*octrineBadgeRepository* té com a dependència el component *EntityManager*. El *BadgeResourceDataTransformer* té com a dependències el *UserResourceDataTransformer* i *ImageResourceDataTransformer*. Finalment, el *ImageResourceDataTransformer* té com a dependència el servei *DiskImageManager*. **Si la creació del *GetBadgeCommandHandler* amb les mateixes dependències es realitza en diferents punts del sistema, seria molt interessant poder reutilitzar el codi d'una forma senzilla, ràpida i eficaç**.

El service container ens ajuda a tractar tots els components software com a serveis i facilita el procés d'instanciació d'objectes. Per a poder definir les classes com a serveis, s'utilitzen fitxers de configuració, en el cas del projecte, fitxers de tipus xml. En aquests fitxers s'assigna un identificador de servei a una classe per a què es pugui utilitzar en un fitxer de configuració o des del codi software d'un component de la capa App.

Els fitxers de configuració es troben localitzats dins la ruta GamificationCoreBundle/Resources/config/container.

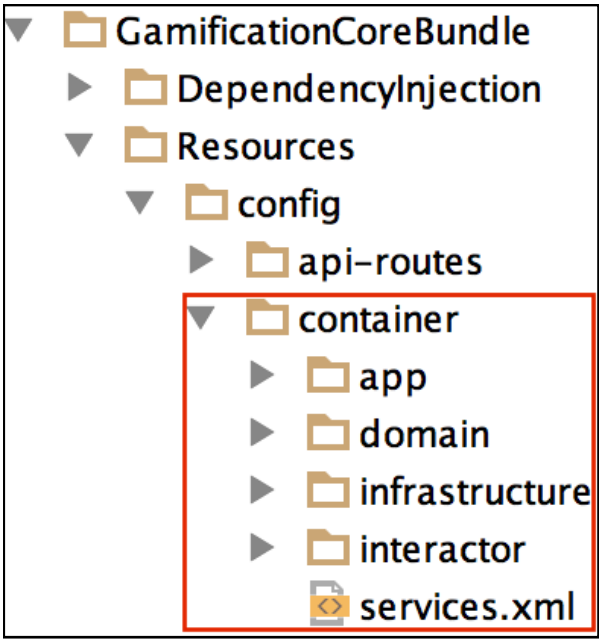


Figura 3 7.1.7.2 Service Container directoris

S'il·lustra la configuració del service container amb el cas de la creació d'un objecte de tipus *GetBadgeCommandHandler*.

En primer lloc, es configuren les dependències les quals no depenen d'altres components software. En aquest cas, les dependències són el *DiskImageManager*, el *EntityManager i* el *UserResourceDataTransformer.*

El *DiskImageManager* està configurat dins el fitxer de configuració domain-services.xml dins la ruta container/infraestructure/domain-services.

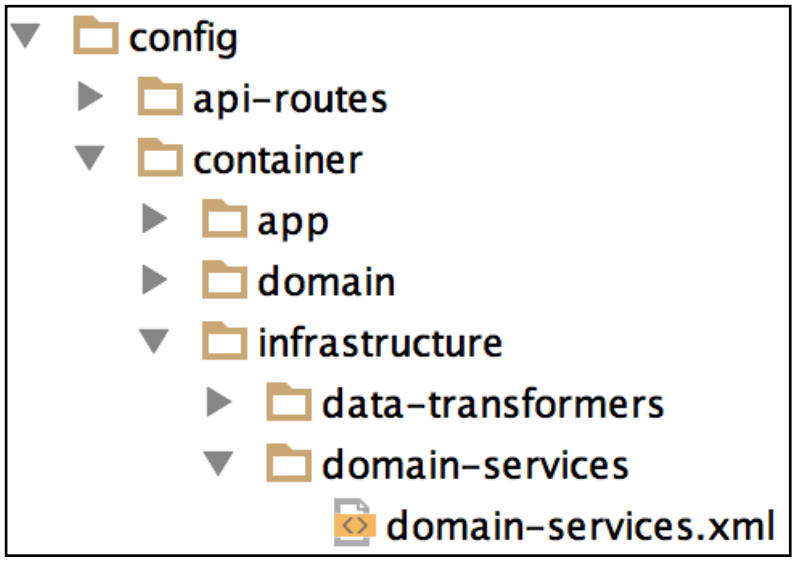


Figura 4 7.1.7.2 Directoris Domain Services Infraestructure

```xml
<?xml version="1.0" ?>

<container xmlns="http://symfony.com/schema/dic/services"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://symfony.com/schema/dic/services http://symfony.com/schema/dic/services/services-1.0.xsd">

    <services>
        <service id="gamification.infrastructure.services.domain.id_generator.uu_id_generator"
                 class="Infrastructure\Services\Domain\IdGenerator\UuIdGenerator">
        </service>

        <service id="gamification.infrastructure.services.domain.password_cipher.md5_password_cipher"
                 class="Infrastructure\Services\Domain\PasswordCipher\Md5PasswordCipher">
        </service>

        <service id="gamification.infrastructure.services.domain.image_manager.disk_image_manager"
                 class="Infrastructure\Services\Domain\ImageManager\DiskImageManager">
        </service>
    </services>

</container>
```

Figura 5 7.1.7.2 Configuració DiskImageManager com un servei

Se li assigna l'identificador de servei a la classe DiskImageManager *gamification.infrastructure.services.domain.image_manager.disk_image_man*

ager el qual coincideix amb la ruta de directoris amb l'arrel el directori Infraestructure on es troba localitzada la classe. Aquest conveni es manté per a tots els identificadors dels serveis.

El component EntityManager de doctrine ja està configurat com a servei per part de la llibreria Doctrine amb l'identificador doctrine.orm.entity_manager.

El component *UserResourceDataTransformer* està configurat en el fitxer data-transformers.xml el qual es troba ubicat en la ruta container/infraestructure/data-transformers amb l'identificador de servei *gamification.infrastructure.data_transformer.resource.domain.entity.user.user_resource_data_transformer*.

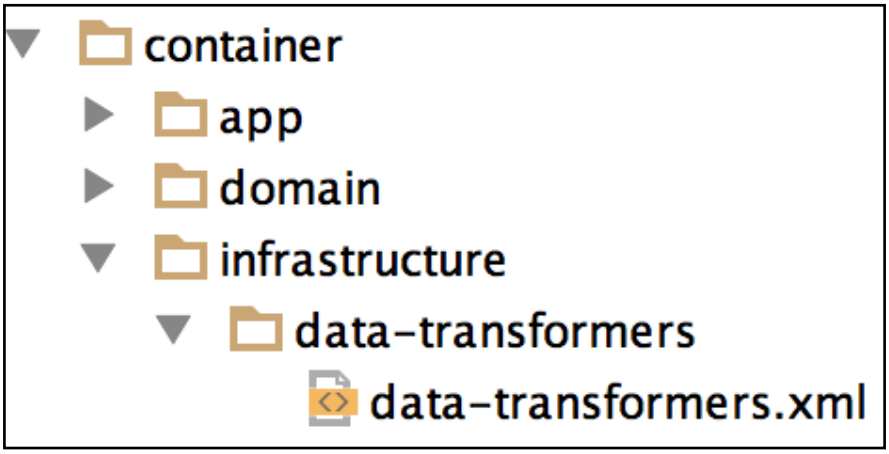


Figura 6 7.1.7.2 Directoris Data Transformers Infraestructure

```
<service id="gamification.infrastructure.data_transformer.resource.domain.entity.user.user_resource_data_transformer"
         class="Infrastructure\DataTransformer\Resource\Domain\Entity\User\UserResourceDataTransformer">
</service>
```

Figura 7 7.1.7.2 Configuració UserResourceDataTransformer

A continuació, es configuren aquells serveis que tenen com a dependència els serveis que s'acaben de configurar. El *ImageResourceDataTransformer* es configura en el mateix fitxer que el *UserResourceDataTransformer*. Per a poder injectar la dependència del *DiskImageManager*, en la seva configuració li passa com a argument l'identificador del servei *DiskImageManager*. L'identificador del ImageResourceDataTransformer és

gamification.infrastructure.data_transformer.resource.domain.entity.image.image_resource_data_transformer.

```
<service id="gamification.infrastructure.data_transformer.resource.domain.entity.image.image_resource_data_transformer"
         class="Infrastructure\DataTransformer\Resource\Domain\Entity\Image\ImageResourceDataTransformer">
    <argument type="service" id="gamification.infrastructure.services.domain.image_manager.disk_image_manager" />
</service>
```

Figura 8 7.1.7.2 Configuració ImageResourceDataTransformer amb dependència de DiskImageManager

A continuació, es configuren les dependències del *GetBadgeCommandHandler*. El *BadgeResourceDataTransformer* es configura en el mateix fitxers que els anteriors DataTransformers que s'han configurat, els quals rep com a dependències. L'identificador del servei és *gamification.infrastructure.data_transformer.resource.domain.entity.badge.badge_resource_data_transformer*.

```
<service id="gamification.infrastructure.data_transformer.resource.domain.entity.badge.badge_resource_data_transformer"
         class="Infrastructure\DataTransformer\Resource\Domain\Entity\Badge\BadgeResourceDataTransformer">
    <argument type="service" id="gamification.infrastructure.data_transformer.resource.domain.entity.image.image_resource_data_transformer" />
    <argument type="service" id="gamification.infrastructure.data_transformer.resource.domain.entity.user.user_resource_data_transformer" />
</service>
```

Figura 9 7.1.7.2 Configuració BadgeResourceDatatransformer amb les seves dependències

El *DoctrineBadgeRespository* està configurat en la ruta container/infrastructure/persistence/repositories/doctrine-repositories en el fitxer doctrine-repositories.xml. Té com a dependència el *EntityManager* i té l'identificador de servei *gamification.infrastructure.peristence.domain.entity.badge.doctrine_badge_repository*.

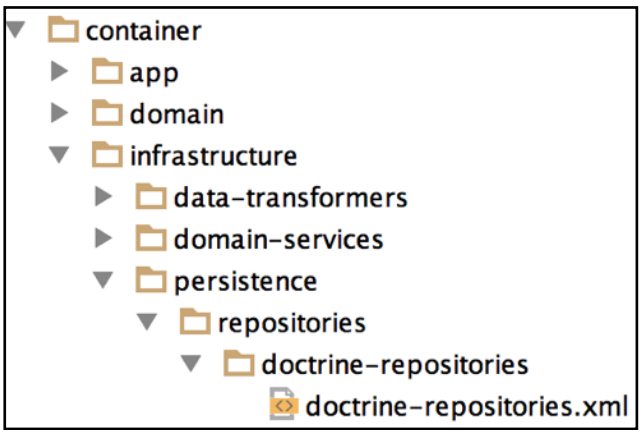


Figura 10 7.1.7.2 Directoris Doctrine Repositories

```
<service id="gamification.infrastructure.peristence.domain.entity.badge.doctrine_badge_repository"
         class="Infrastructure\Persistence\Doctrine\Domain\Entity\Badge\DoctrineBadgeRepository">
    <argument type="service" id="doctrine.orm.entity_manager" />
</service>
```

Figura 11 7.1.7.2 Configuració DoctrineBadgeRespoitory amb la dependència EntityManager

Tot seguit, **es creen alies per a mantenir l'abstracció dels components d'infraestructura amb els components de domini**. Només té efectes en l'àmbit de nomenclatura. Malgrat això, atorga de poder semàntic i es fa la configuració més intel·ligible. Els alies de la capa Domain estan ubicats en la ruta container/app/domain. Els alies dels DataTransformers, estan localitzats en el directori data-transformers en el fitxer data-transformers.xml i el dels Respositories en la carpeta repositories en el fitxer repositories.xml.

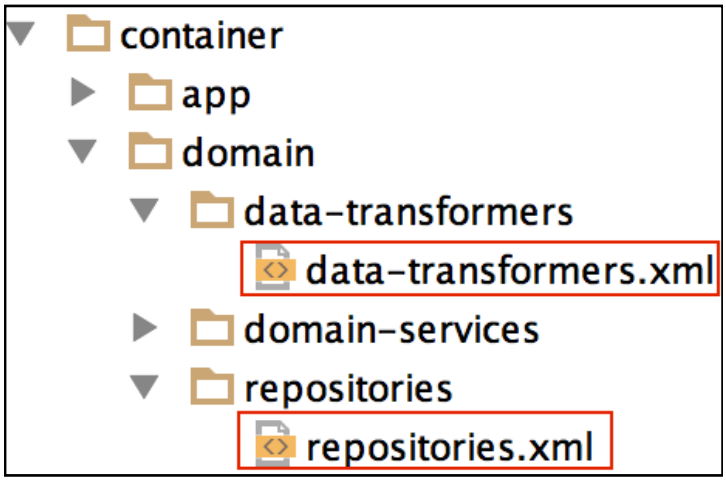


Figura 12 7.1.7.2 Fitxers per alies de Domain

D'aquesta forma, el servei *DoctrineBadgeRepository* té com identificador de servei *gamification.domain.entity.badge.badge_repository*, fent referència a la interfície de domini. De la mateixa forma, el *BadgeResourceDataTransformer* té com identificador de servei *gamification.domain.entity.badge.badge_data_transformer.*

```
<service id="gamification.domain.entity.badge.badge_repository"
         alias="gamification.infrastructure.peristence.domain.entity.badge.doctrine_badge_repository" />
```

Figura 13 7.1.7.2 Alies BadgeRespository pel DoctrineBadgeRepository

```
<service id="gamification.domain.entity.badge.badge_data_transformer"
         alias="gamification.infrastructure.data_transformer.resource.domain.entity.badge.badge_resource_data_transformer" />
```

Figura 14 7.1.7.2 Alies BadgeDataTransformer pel BadgeResourceDataTransformer

Finalment, es configura la classe *GetBadgeCommandHandler* com a servei. La configuració dels CommandHandlers com a serveis es troben en la ruta de directoris container/interactor/command-handlers en el fitxer command-handlers.xml. Es configura el CommandHandler passant-li com a paràmetre els alies del *BadgeRepository* i del *BadgeDataTransformer.* Té com identificador de servei *gamification.interactor.command_handler.get_badge.get_badge_command_handler*.

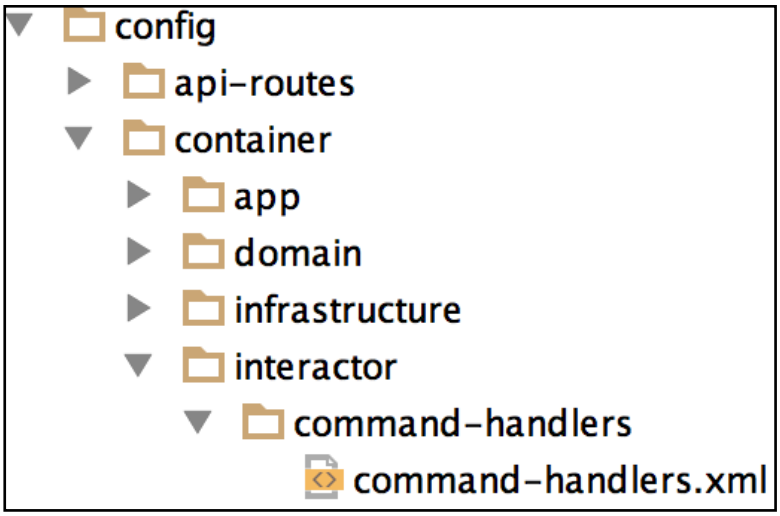


Figura 15 7.1.7.2 Directoris configuració CommandHandlers

```
<service id="gamification.interactor.command_handler.get_badge.get_badge_command_handler"
         class="Interactor\CommandHandler\GetBadge\GetBadgeCommandHandler">
    <argument type="service" id="gamification.domain.entity.badge.badge_repository" />
    <argument type="service" id="gamification.domain.entity.badge.badge_data_transformer" />
</service>
```

Figura 16 7.1.7.2 Configuració GetBadgeCommandHandler

Amb la utilització del service container, el codi per instanciar el *GetBadgeCommandHandler* en qualsevol punt del codi del sistema, es redueix a obtenir a través del service container el CommandHandler mitjançant l'identificador de servei que té assignat. Es

tracta d'una estratègia molt útil amb la qual es reutilitza codi i es poden modificar les dependències dels CommandHandlers d'una forma àgil i amb poc impacte en el codi, facilitant el manteniment de l'aplicació.

```
$getBadgeCommandHandler = $this->get(
    "gamification.interactor.command_handler.get_badge.get_badge_command_handler"
);
```

Figura 17 7.1.7.2 Instanciar GetBadgeCommandHandler com a servei

7.2. API REST

En aquest punt del procés d'implementació del sistema, ja es disposa de tots els elements necessaris per a iniciar la creació de la **API REST** i orientar el sistema cap a l'**arquitectura de tipus Software as a Service (SaaS)**. **Els elements que conformaran l'API estaran ubicats en la capa App**.

7.2.1. Concepte

REST [51] (REpresentational State Transfer) és un tipus d'**arquitectura de desenvolupament web que es basa totalment en l'estàndard HTTP i es tracta del tipus d'arquitectura més natural i estàndard per a crear APIs per a serveis orientats a Internet**. Aquest tipus d'arquitectura, **ens permet crear serveis i aplicacions les quals poden ser utilitzades per qualsevol dispositiu o client que sigui capaç d'interpretar el protocol HTTP**. El protocol HTTP [52] (Hypertext Transfer Protocol) és un protocol de nivell capa d'aplicació sense estat, el qual s'utilitza per comunicar sistemes distribuïts. Aquest protocol ha contribuït en la creació de la web actual tal com la coneixem. Gairebé la totalitat dels llocs online, utilitzen aquest protocol per servir els seus continguts als clients o altres sistemes.

En aquest tipus d'arquitectura, el servidor que conté l'API REST exposa els **endpoints** els quals són els punts d'entrada per a què els clients es puguin connectar al sistema. **Solen ser de tipus URL i indiquen a quins tipus de recursos i de quina forma poden accedir els clients del sistema**. El servidor enviarà una **resposta als clients en format determinat**

(json, xml, hateoas), prèviament pactat amb el client, **i un codi d'estat HTTP** [53] el qual indicarà l'estat de l'execució.

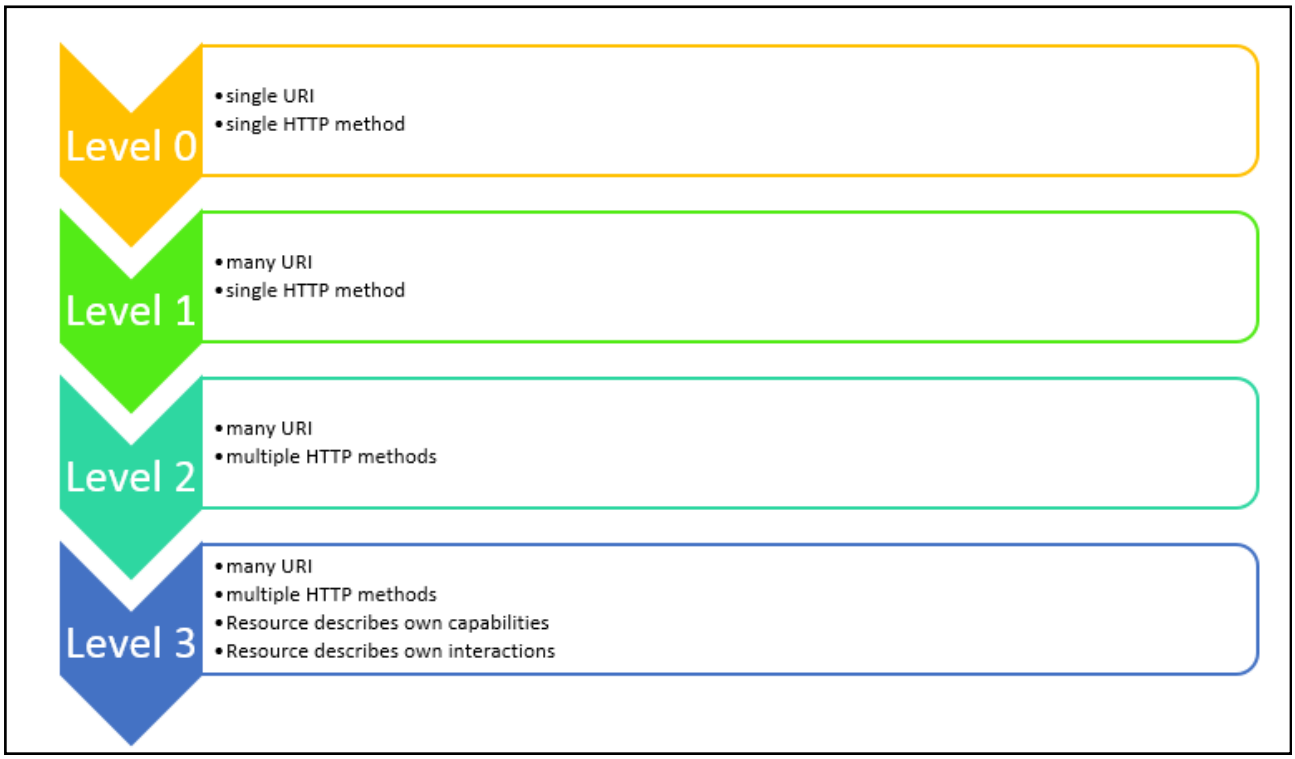


Figura 1 7.2.2 Descripció nivells API REST

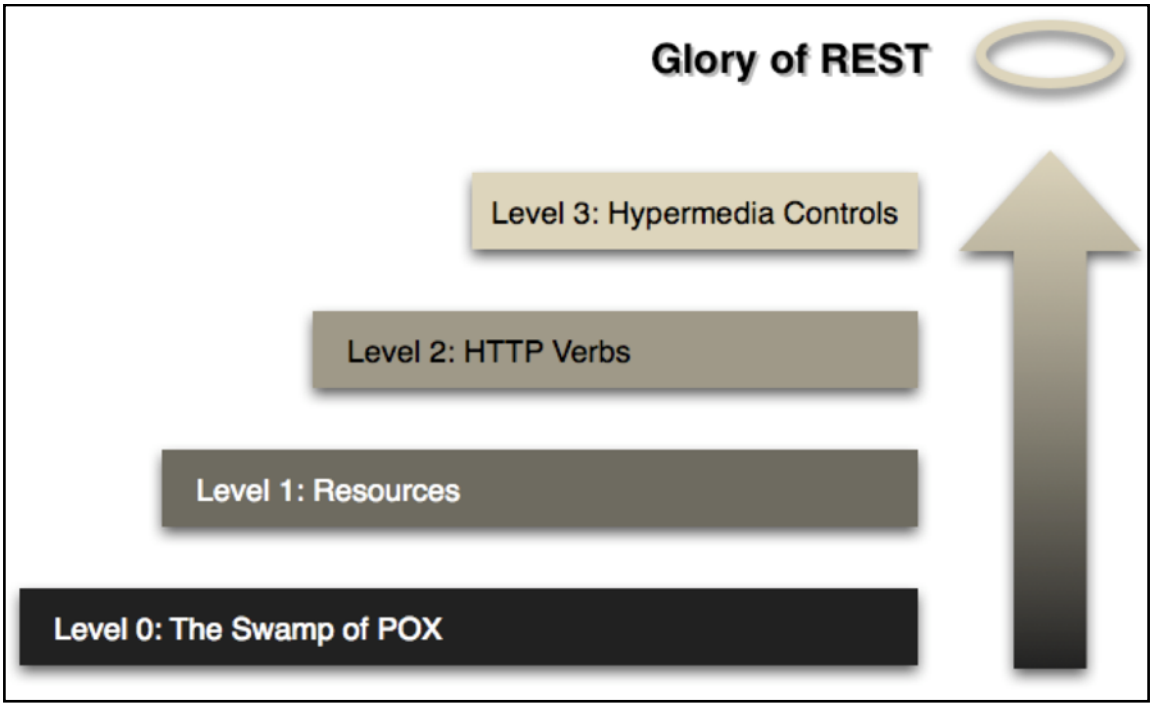


Figura 2 7.2.2 Nivells API REST

7.2.2. Nivells API REST

Una API REST es pot definir en diferents **nivells** segons el seu estat de maduresa i de complexitat. Aquest tipus de divisió es defineix com el *Richardson Maturity Model* [54][55]. L'API REST desenvolupada en aquest projecte té el nivell 3 Hypermedia que és el màxim nivell que es pot assolir.

7.2.2.1. Nivell 0

Aquest nivell només **utilitza el protocol HTTP per a intercanvi d'informació** entre el client i el servidor mitjançant un endpoint determinat.

No s'utilitzen codis d'estat HTTP. En el cas que per exemple l'execució hagi finalitzat amb error per alguna causa, el servidor respondrà amb un codi d'estat HTTP 200 OK, el qual significa que la transacció ha finalitzat correctament. El servidor pot adjuntar un missatge descriptiu sobre l'error el qual el client haurà d'entendre per a poder discernir si s'ha produït un error o si l'execució ha finalitzat amb èxit.

Només s'utilitza un tipus de verb HTTP el qual sol ser el mètode **GET**. Els verbs HTTP [56] solen indicar al servidor quin tipus d'acció es vol realitzar sobre un determinat recurs.

7.2.2.2. Nivell 1: Resources

En aquest nivell, **cada funcionalitat del sistema té el seu propi endpoint / resource (multiple resources)**. D'aquesta forma, la resposta de cada resource s'utilitza per invocar un altre resource de la API. No obstant el client ha de conèixer prèviament com accedir al següent endpoint.

El client segueix utilitzant **un únic verb HTPP** i el servidor **respon amb un únic codi d'estat HTTP**.

7.2.2.3. Nivell 2: HTTP Verbs

El **client es comunica amb l'API REST utilitzant diferents verbs HTTP**. Els més utilitzats són:

- **GET**
  Per obtenir informació.
  Els paràmetres es codifiquen en la URL del resource.
- **POST**
  Per a crear informació.
  Els paràmetres es codifiquen en el cos del missatge.
- **PUT**
  Actualitza la informació.
  Els paràmetres es codifiquen en el cos del missatge.
- **DELETE**
  S'esborra informació.
  Els paràmetres es codifiquen en el cos del missatge.

El servidor respon al client amb **diferents codis d'estat HTTP** segons l'estat de l'execució.

7.2.2.4. Nivell 3: Hypermedia

Els recursos / endpoints afegeixen dos nous elements a la resposta per part del servidor:

- **Hypermedia link al propi resourse / endpoint**
- **Hypermedia link a resources / endpoints relacionats**

D'aquesta forma, **els resources / endpoints s'autodescubreixen al client ells mateixos i, adicionalment, indiquen al client funcionalitats relacionades que pot utilitzar**. El client és capaç en aquest context de navegar pel sistema sense necessitat de tenir coneixement previ de totes les funcionalitats que ofereix l'API REST. És com una forma de documentació sota demanda. Com més el client utilitza l'API, més descobreix les funcionalitats que aquesta ofereix i que pot utilitzar. Una bona analogia seria la d'un usuari en una pàgina web i els links que aquesta ofereix, els quals estan relacionats a altres pàgines web.

Aquest concepte s'anomena **HATEOAS** [57].

7.2.3. EndPoints

Aquests són els endpoints de l'API REST implementats actualment. El format utilitzat per la comunicació entre el client i el servidor és **JSON + HAL**. JSON [58] (Javascript Object Notation) és una codificació lleugera molt estesa i utilitzada en la comunicació web entre diferents sistemes. HAL [59] (Hypertext Application Language) és la codificació per a incluir els resources relacionats en la resposta d'un endpoint en concret per a API REST de nivell 3.

Es descriu per cada un d'ells la següent informació:

- Funcionalitat
- URL de l'endropint
- Verb HTTP
- Format
- Paràmetres d'entrada
- Codis d'estat HTTP
- Resposta

UserSignUp

- Funcionalitat: Crear un nou usuari
- URL: https://badges-io-dev/api/user/signup
- Verb HTTP: POST
- Format: HAL + JSON
- Paràmetres d'entrada: username, email, password
- Codis d'estat HTTP: 200, 400, 500
- Resposta: Valors atributs user i resources relacionats

UserLogIn

- Funcionalitat: Accés d'un usuari al sistema
- URL: https://badges-io-dev/api/user/login
- Verb HTTP: PUT
- Format: HAL + JSON
- Paràmetres d'entrada: username // email, password
- Codis d'estat HTTP: 200, 400, 404, 403, 500
- Resposta: Valors atributs user i resources relacionats

BadgeCreate

- Funcionalitat: Crear un nou badge i una nova imatge
- URL: https://badges-io-dev/api/badge/create
- Verb HTTP: POST
- Format: HAL + JSON
- Paràmetres d'entrada: name, description, userId, isMultiUser, imageName, imageWidth, imageHeight, imageFormat, imageFile
- Codis d'estat HTTP: 200, 400, 404, 500
- Resposta: Valors atributs badge, image, user i resources relacionats

BadgeUpdate

- URL: https://badges-io-dev/api/badge/update
- Verb HTTP: POST
- Format: HAL + JSON
- Paràmetres d'entrada: id, name, description, userId, isMultiUser, imageName, imageWidth, imageHeight, imageFormat, imageFile
- Codis d'estat HTTP: 200, 400, 404, 401, 500
- Resposta: Valors atributs badge, image, user i resources relacionats

BadgeGet

- URL: https://badges-io-dev/api/badge/{badeId}/{userId}
- Verb HTTP: GET
- Format: HAL + JSON
- Paràmetres d'entrada: id, userId
- Codis d'estat HTTP: 200, 400, 404, 401, 500
- Resposta: Valors atributs badge, image, user i resources relacionats

BadgeList

- URL: https://badges-io-dev/api/badge/list/{userId}
- Verb HTTP: GET
- Format: HAL + JSON
- Paràmetres d'entrada: userId
- Codis d'estat HTTP: 200, 400, 404, 500
- Resposta: Col·lecció de valors atributs badge, image, user i resources relacionats

BadgeDelete

- URL: https://badges-io-dev/api/badge/{badeId}/{userId}
- Verb HTTP: DELECTE
- Format: HAL + JSON
- Paràmetres d'entrada: id, userId
- Codis d'estat HTTP: 200, 400, 404, 401, 500
- Resposta: Missatge descriptiu estat acció i resources relacionats

7.2.4. Symfony API REST

Un cop s'han descrit de forma genèrica els diferents endpoints, en aquest apartat s'explica com s'han implementat.

Els components principals són els ***Controllers*** els quals es troben ubicats en l'arquitectura en la **capa App**. Aquests components seran els responsables **d'oferir un GUI** a l'usuari per a què aquest pugui interactuar amb el sistema, **processaran la informació que els proporciona a l'usuari**, **invocaran el CommandHandler a través del service container i mostraran el resultat de l'execució a l'usuari**.

Com ja s'ha comentat, el framework Symfony encapsula les funcionalitats en bundles. Els *Controllers*, els quals defineixen els endpoints de l'API REST, es troben ubicats en la ruta App/Bundle/GamificationApiBundle/Controller. En aquest directori es troben el *BadgeApiController*, el qual implementa els endpoints que gestionen els *Badges*, i el *UserApiController*, el qual implementa els endpoints que gestionen els *Users*.

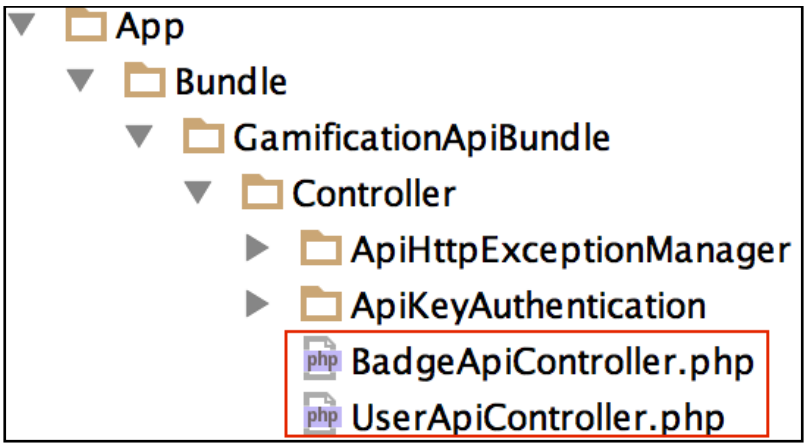


Figura 1 7.2.4 API controllers

Per implementar els controllers, s'ha utilitzat el Bundle o llibreria externa **FOSRestBundle** [60] el qual facilita la implementació i la creació d'APIs REST mitjançant el framework Symfony.

A continuació i a manera d'exemple, s'explica com s'ha implementat el endpoint **BadgeGet**.

```
/**
 * @Get("/badge/{id}/{userId}")
 */
public function getBadgeAction($id, $userId)
{
    try {
        $getBadgeCommand = $this->buildGetBadgeCommandByRequest($id, $userId);

        return $this->container->get(
            'gamification.interactor.command_handler.get_badge.get_badge_command_handler'
        )->handle($getBadgeCommand);
    } catch (\Exception $applicationException) {
        throw $this->buildBadgeHttpExceptionManager()
            ->applicationGetBadgeExceptionToHttpException($applicationException);
    }
}
```

Figura 2 7.2.4 Endpoint BadgeGet

Els *Controllers* estan compostos per *Actions*. Aquests són mètodes públics dins del *Controller* els quals el nom del mètode acaba amb el sufix Action. Adicionalment, per a crear controllers de tipus *FOSRestBundle*, s'afegeix al nom del mètode el nom del verb HTTP el qual serà utilitzat per part del client per invocar el endpoint com a prefix. La annotation @Get("urlRelativa") que ofereix el *FOSRestBundle* serveix per a generar la URL que serà utilitzada per invocar l'action. D'aquesta forma, pel

cas **BadgeGet** es crea un mètode públic amb el nom de **getBadgeAction** i la annotacion **@Get("/badge/{id}/{userId}")** dins el *BadgeApiController*. Els paràmetres id i userId els quals formen part de la URL, seran **paràmetres d'entrada** pel mètode **getBadgeAction**.

El *getBadgeAction* invoca el mètode privat *buildGetBadgeCommandByRequest* amb els paràmetres d'entrada id i userId.

```
private function buildGetBadgeCommandByRequest($id, $userId)
{
    return new GetBadgeCommand($id, $userId);
}
```

Figura 3 7.2.4 Mètode buildGetBadgeCommandByRequest

Aquest mètode crearà un objecte de tipus *GetBadgeCommand* passant-li com a paràmetres el id i el userId. A continuació, el mètode *getBadgeAction* instanciarà la classe *GetBadgeCommandHandler* mitjançant el service container passant-lo com a paràmetre l'identificador del servei. Finalment, invocarà el mètode *handle* del *CommandHandler* passant-li com paràmetre d'entrada el *Command* acabat de crear i mostrarà el resultat de l'execució a l'usuari.

En el cas que hagi algun error durant l'execució, es capturarà l'error de la capa *Interactor* i es convertirà en un error de tipus HTTP amb el seu codi d'estat associat. Els components els quals realitzen aquesta funció són els *HttpExceptionManager*. Aquests estan ubicats dins de la carpeta GamificationApiBundle/Controller/ApiHttpExceptionManager. Aquests components estan declarats com a serveis en el service container. Existeix un per a gestionar les excepcions de cada controller.

D'aquesta forma, les excepcions *Interactor* capturades pels actions del *BadgeApiController*, les convertirà en excepcions de tipus HTTP el *BadgeApiHttpExceptionManager* i les del *UserApiController* les gestionarà el *UserApiHttpExceptionManager*.

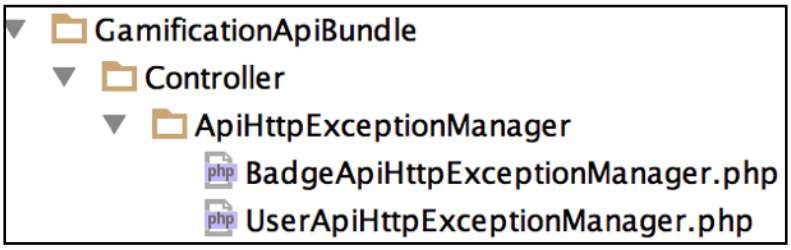


Figura 4 7.2.4 ApiHttpExceptionManagers directoris

D'aquest forma, el *getBadgeAction* captura l'excepció i invoca el mètode privat *buildBadgeHttpExceptionManager*, el qual instanciarà a través del service container la classe *BadgeApiHttpExceptionManager*.

```php
/**
 * @return BadgeApiHttpExceptionManager
 */
private function buildBadgeHttpExceptionManager()
{
    // @codingStandardsIgnoreStart
    return $this->container->get(
        'gamification_api_bundle.controller.api_http_exception_manager.badge_api_http_exception_manager'
    );
    // @codingStandardsIgnoreEnd
}
```

Figura 5 7.2.4 Mètode buildBadgeHttpExceptionManager

```php
/**
 * @param \Exception $applicationException
 *
 * @return HttpException
 */
public function applicationGetBadgeExceptionToHttpException(\Exception $applicationException)
{
    if ($applicationException instanceof InvalidGetBadgeCommandException) {
        $statusCode = Response::HTTP_BAD_REQUEST;
    } elseif ($applicationException instanceof InvalidGetBadgeCommandHandlerException) {
        $statusCode = static::GET_BADGE_MAP_HTTP_CODE_EXCEPTION[$applicationException->getCode()];
    } else {
        $statusCode = Response::HTTP_INTERNAL_SERVER_ERROR;
    }

    return $this->buildHttpException($applicationException, $statusCode);
}
```

Figura 6 7.2.4 applicationGetBadgeExceptionToHttpException mètode

D'aquest forma, el *getBadgeAction* captura l'excepció i invoca el mètode privat *buildBadgeHttpExceptionManager*, el qual instanciarà a través del service container la classe *BadgeApiHttpExceptionManager*.

```
const GET_BADGE_MAP_HTTP_CODE_EXCEPTION = [
    InvalidGetBadgeCommandHandlerExceptionCode::STATUS_CODE_BADGE_NOT_FOUND =>
        Response::HTTP_NOT_FOUND,
    InvalidGetBadgeCommandHandlerExceptionCode::STATUS_CODE_USER_FORBIDDEN =>
        Response::HTTP_UNAUTHORIZED
];
```

Figura 7 7.2.4 Mappegi Interactor exception Http exception

Per finalitzar la gestió de conversió d'errors, s'invocà al mètode privat *buildHttpException* de la classe *BadgeApiHttpExceptionManager.*

```
/**
 * @param \Exception $applicationException
 * @param string $statusCode
 *
 * @return HttpException
 */
private function buildHttpException(\Exception $applicationException, $statusCode)
{
    return new HttpException($statusCode, $applicationException->getMessage());
}
```

Figura 7 7.2.4 Mètode buildHttpException

Finalment i per crear la resposta i mostrar-la a l'usuari, s'ha utilitzat un segon bundle o llibreria de tercers anomenada **JMSSerializer** [61]. Aquest bundle servirà de suport per a poder crear les respostes amb el format JSON + HAL. Mitjançant fitxers de configuració de tipus yml, la llibreria permet associar una classe PHP per a transformar-la en format JSON + HAL, definir quins camps es mostren i definir els resources associats així com els objectes embedded. Els fitxers de configuració es troben ubicats en la ruta Infrastructure/Resource/Domain/Jms/Resources/serializer.

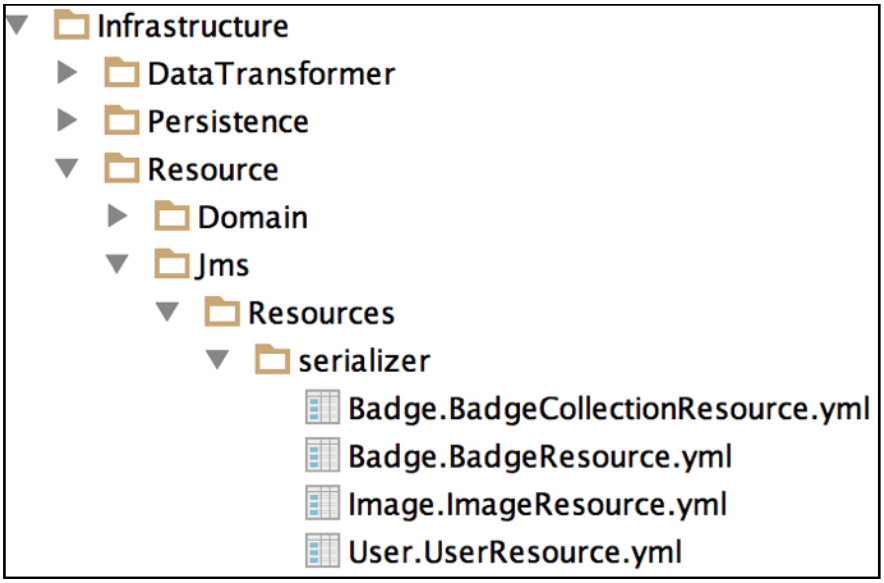


Figura 8 7.2.4 JMSSerializer fitxers de configuració directori

En aquests fitxers **es configura els objectes de tipus *Resource* que creen els *DataTransformers* per a què se serialitzin en format de tipus JSON + HAL**. El procés de serialització és aquell procés que transforma un objecte en un format determinat.

A continuació i a manera d'exemple, es descriu el fitxer de configuració per serialitzar l'objecte *BadgeResource.*

El fitxer de configuració del *BadgeResource* és el *Badge.BadgeResource.yml*.

```
Infrastructure\Resource\Domain\Entity\Badge\BadgeResource:
    exclusion_policy: ALL
    xml_root_name: badge
```

Figura 9 7.2.4 Definició classe en el serialitzar

En primer lloc es defineix quina classe es vol configurar per a serialitzar, en aquest exemple és el *BadgeResource*. També es configura la política d'exposició dels atributs. Aquesta política, definirà quins atributs de la classe seran serialitzats i es mostraran en la resposta. En el nostre cas, s'exclouen tots els atributs a ser exposats per defecte. D'aquesta forma, els atributs que es vulguin exposar s'hauran indicat de forma explícita en el fitxer de configuració. El xml_root_name servirà per afegir els prefixos als elements embedded de la resposta.

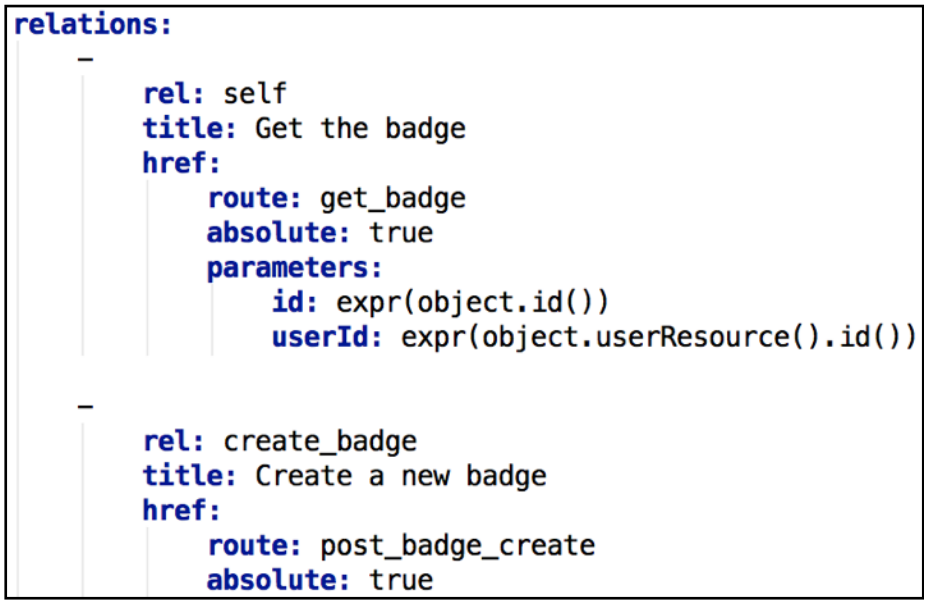
A continuació, es defineixen en l'apartat de *properties* els camps de la classe que es volen exposar així com el seu tipus. En aquest cas exposem tots els atributs simples de la classe *BadgeResource*.

```
properties:
    id:
        expose: true
        type: string

    name:
        expose: true
        type: string

    description:
        expose: true
        type: string

    isMultiUser:
        expose: true
        type: boolean
```

Figura 10 7.2.4 Atributs a exposar

Finalment, es defineix l'apartat de *relations*. N'hi ha de dos tipus: de tipus href i de tipus embedded.

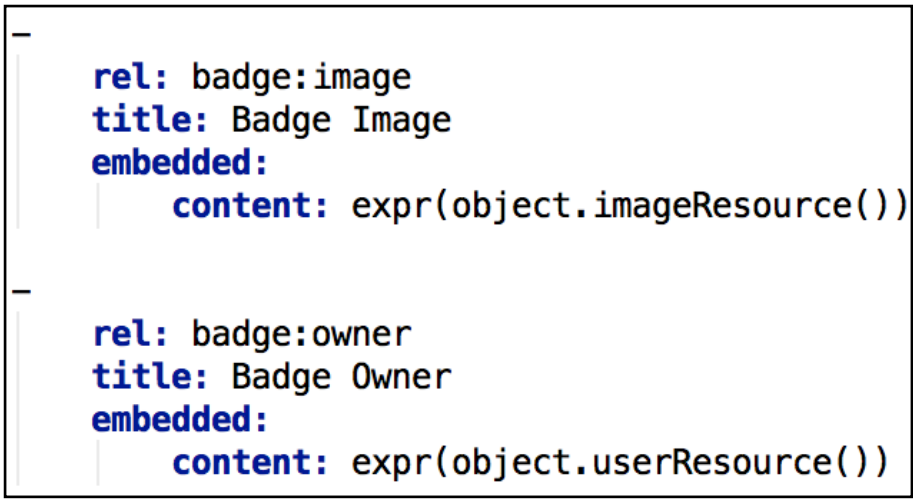
Els relations de tipus href defineixen links resources / endpoints de l'api relacionats. En la imatge següent es mostra alguns dels endpoints relacionats amb el *BadgeResource*. El primer és el sefLink, el qual és el link del endpoint per obtenir la informació del resource en qüestió. En aquest cas és l'endpoint *BadgeGet* al qual li passen per paràmetres l'id del badge i l'id del user. Aquesta configuració, crearà una URL absoluta per a poder accedir al BadgeGet endpoint amb tots els seus paràmetres. Poden existir altre tipus de relacions que no siguin self. En aquest cas es configura també l'accés a l'endpoint BadgeCreate.



```
relations:
    -
        rel: self
        title: Get the badge
        href:
            route: get_badge
            absolute: true
            parameters:
                id: expr(object.id())
                userId: expr(object.userResource().id())

    -
        rel: create_badge
        title: Create a new badge
        href:
            route: post_badge_create
            absolute: true
```

Figura 11 7.2.4 Relations de tipus href

Finalment, les relations de tipus embedded són aquelles que afegeixen a la resposta altres objectes que tenen relació amb l'objecte que s'està serialitzant. En el nostre cas, un objecte de tipus *BadgeResource* està relacionat amb un objecte de tipus *ImageResource* i amb un objecte de tipus *UserResource*. Aquests també tenen els seus propis fitxers de configuració els quals s'utilitzen per incrustar la informació de les classes a la resposta.



```
-
    rel: badge:image
    title: Badge Image
    embedded:
        content: expr(object.imageResource())

-
    rel: badge:owner
    title: Badge Owner
    embedded:
        content: expr(object.userResource())
```

Figura 12 7.2.4 Relations de tipus embedded

A continuació, es mostra una execució de l'endpoint *BadgeGet* amb un badgeId 0314e8b9-62ac-4eba-84d1-240a7f5cfdaf i un userId e4f2c464-c486-4047-86c7-4cab7f7b9866 en un navegador web. Aquesta seria la URL resultant de l'endpoint: https://badges-io-dev/api/badge/0314e8b9-62ac-4eba-84d1-240a7f5cfdaf/e4f2c464-c486-4047-86c7-4cab7f7b9866.


```
{
  id: "0314e8b9-62ac-4eba-84d1-240a7f5cfdaf",
  name: "Meloman",
  description: "Li agrada molt la música",
  is_multi_user: true,
  _links: {
    self: {
      href: "https://badges-io-dev/api/badge/0314e8b9-62ac-4eba-84d1-240a7f5cfdaf/e4f2c464-c486-4047-86c7-4cab7f7b9866"
    },
    create_badge: {
      href: "https://badges-io-dev/api/badge/create"
    },
    delete_badge: {
      href: "https://badges-io-dev/api/badge/0314e8b9-62ac-4eba-84d1-240a7f5cfdaf/e4f2c464-c486-4047-86c7-4cab7f7b9866"
    },
    update_badge: {
      href: "https://badges-io-dev/api/badge/update"
    },
    list_badges: {
      href: "https://badges-io-dev/api/badges/list/e4f2c464-c486-4047-86c7-4cab7f7b9866"
    }
  },
  _embedded: {
    badge:image: {
      id: "0314e8b9-62ac-4eba-84d1-240a7f5cfdaf",
      name: "meloman",
      width: 20,
      height: 20,
      format: "jpg",
      href: "https://badges-io-dev/0314e8b9-62ac-4eba-84d1-240a7f5cfdaf.jpg"
    },
    badge:owner: {
      id: "e4f2c464-c486-4047-86c7-4cab7f7b9866",
      email: "miquel.marino@atrapalo.com",
      username: "miquel.marino",
      _links: {
        login: {
          href: "https://badges-io-dev/api/user/login"
        },
        signup: {
          href: "https://badges-io-dev/api/user/signup"
        },
        list_badges: {
          href: "https://badges-io-dev/api/badges/list/e4f2c464-c486-4047-86c7-4cab7f7b9866"
        },
        create_badge: {
          href: "https://badges-io-dev/api/badge/create"
        }
      }
    }
  }
}
```


Figura 13 7.2.4 Resultat Execució endpoint BadgeGet en un navegador web

7.2.5. Sandbox ApiDoc

Un fet molt important per a garantir la correcta utilització de l'API i aprofitar al màxim el potencial de les funcionalitats que aquesta ofereix per part del client, és el de poder **disposar d'una bona documentació de l'API REST la qual ha d'estar actualitzada a l'última versió**. Sovint és difícil poder mantenir una bona documentació en paper, ja que el programador ha d'estar constantment pendent d'actualitzar la documentació sempre que hi ha un canvi en una funcionalitat o se'n crea una nova. Per una altra banda, seria molt convenient poder oferir al client un entorn online per a què aquest pugui realitzar proves i, a poder ser, que ofereixi una interfície d'usuari amigable i usable.

Per aquestes raons, s'ha integrat al sistema un nou bundle de symfony anomenat **NelmioApiDocBundle** [62]. Aquest bundle, permet generar documentació de la API REST on-the-fly i disponible online mitjançant una URL. En el cas del projecte la URL es https://badges-io-dev/api/doc.

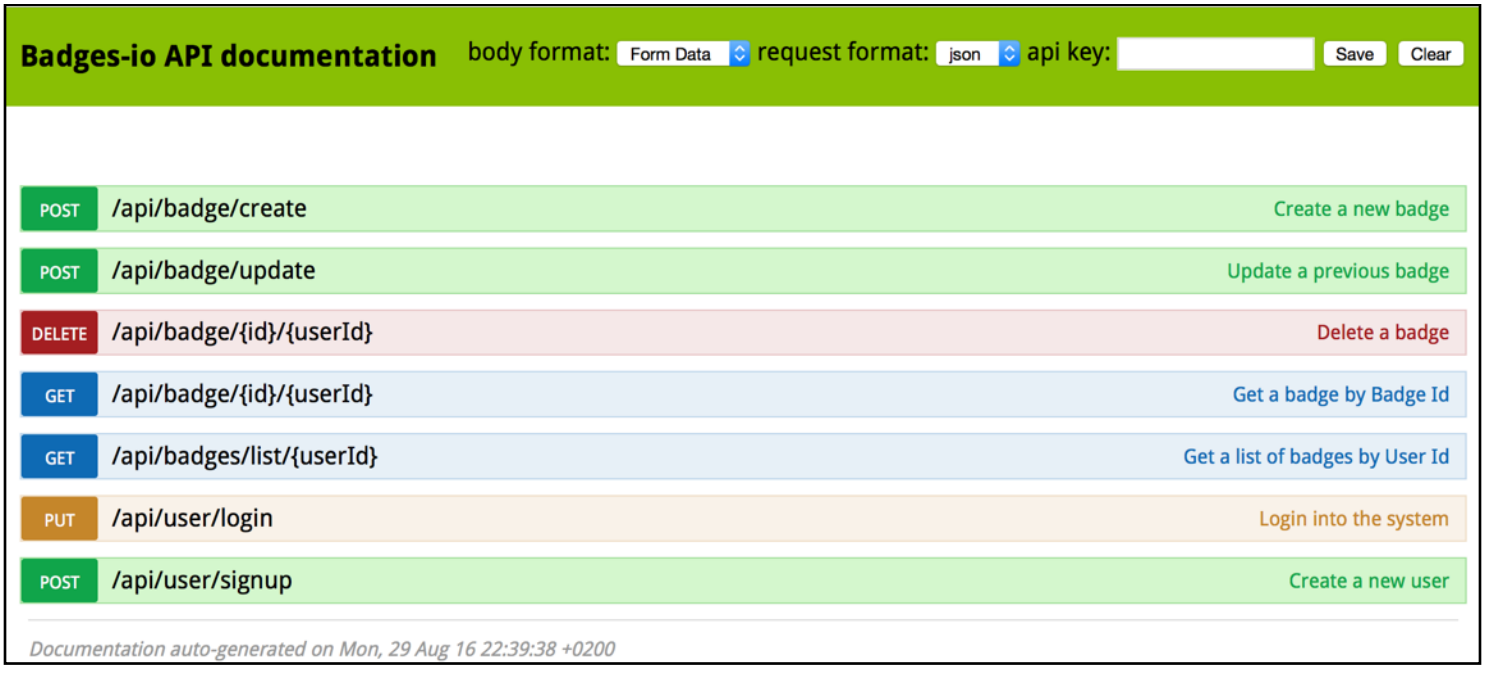
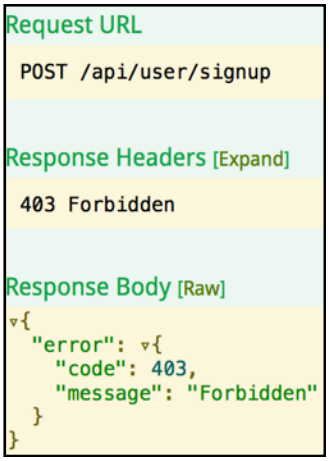


Figura 1 7.2.5 Documentació Endpoints API

A continuació, s'il·lustra una imatge la visualització general dels endpoints que genera el bundle com a **documentació de l'API**.

Com es pot observar, cada endpoint està precedit pel verb HTTP que s'utilitza, així com la URL relativa per a poder executar l'endpoint i una breu descripció.

A la part superior dreta, es pot observar que hi ha un camp anomenat **API key**. L'API key sol ser un valor alfanumèric que serveix per autenticar les respostes i és una de les formes més esteses, combinada amb d'altres, per a validar la comunicació entre client i l'API REST [63]. L'API key del projecte té el valor 8eb7afb9-af81-4667-a629-74009c600ae0. En el cas de què la petició a un endpoint per part del client no informi del valor correcte de l'API key, el sistema retornarà un codi d'estat HTTP de tipus error 403 el qual informa que l'usuari no està autenticat.



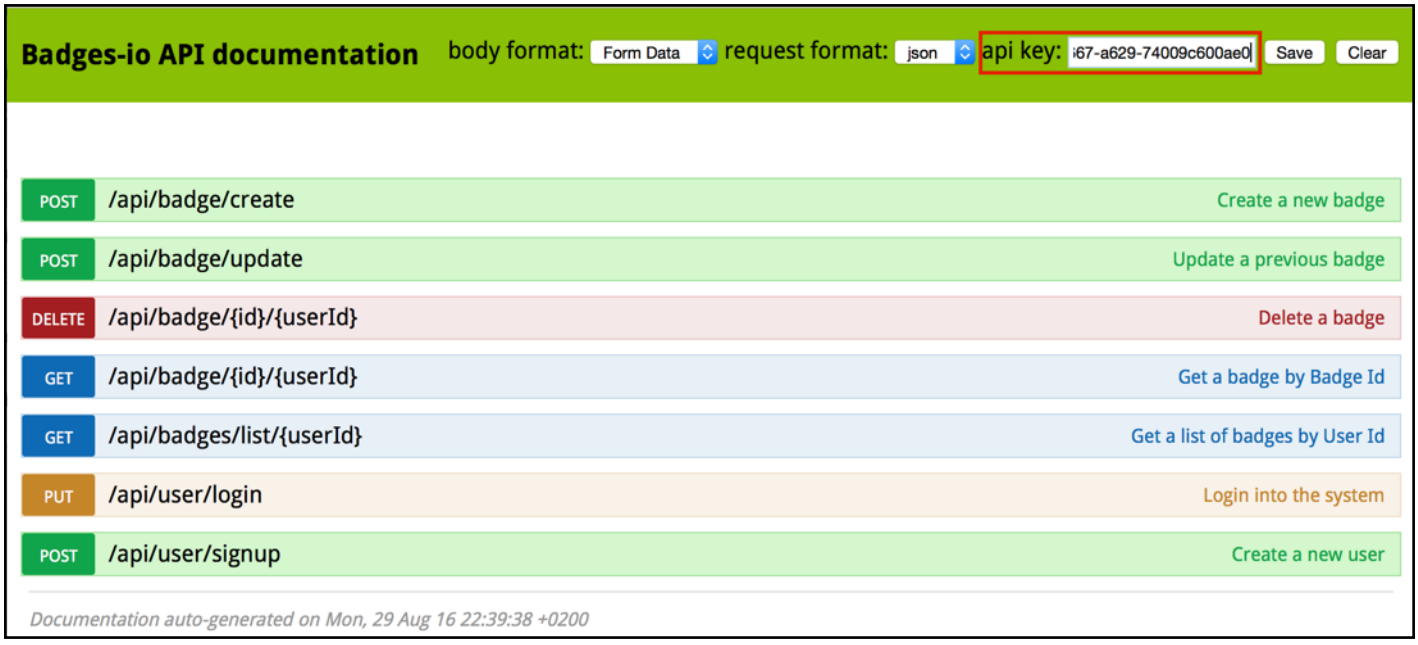


Figura 2 7.2.5 API Key i missatge d'error en cas de que el client no informi del seu valor

Cada endpoint del llistat, si és clicka amb el ratolí sobre un d'ells, es desplega la documentació detallada de cada un d'ells. A continuació, es pot observar la documentació detallada de l'endpoint BadgeList, on es pot veure els paràmetres necessaris per executar l'endpoint i els codis d'estat HTTP que retorna com a resultat de l'execució.

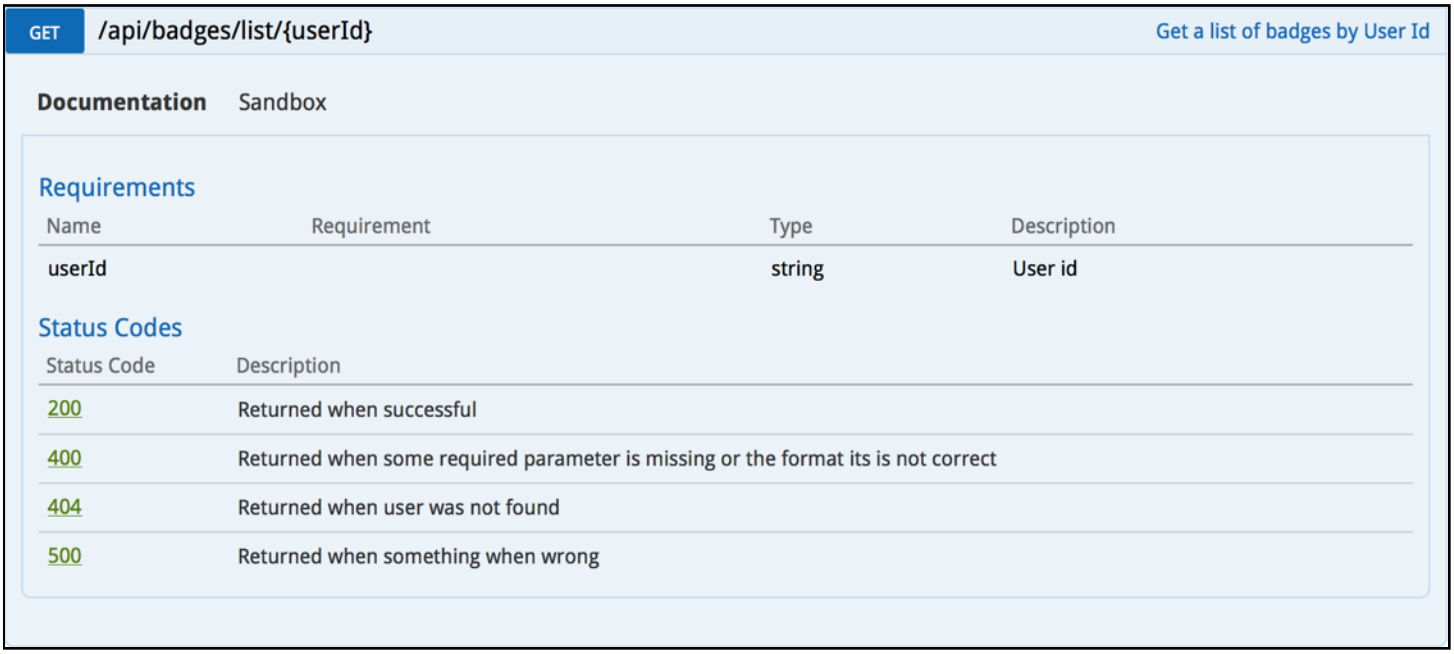


Figura 3 7.2.5 Documentació BadgeList endpoint

A part de la documentació, el *NelmioApiDocBundle* ofereix un entorn de **sandbox** amb una interfície d'usuari per a què el client pugui realitzar proves i testejar el sistema. A continuació, s'il·lustra la funcionalitat amb el endpoint BadgeList.

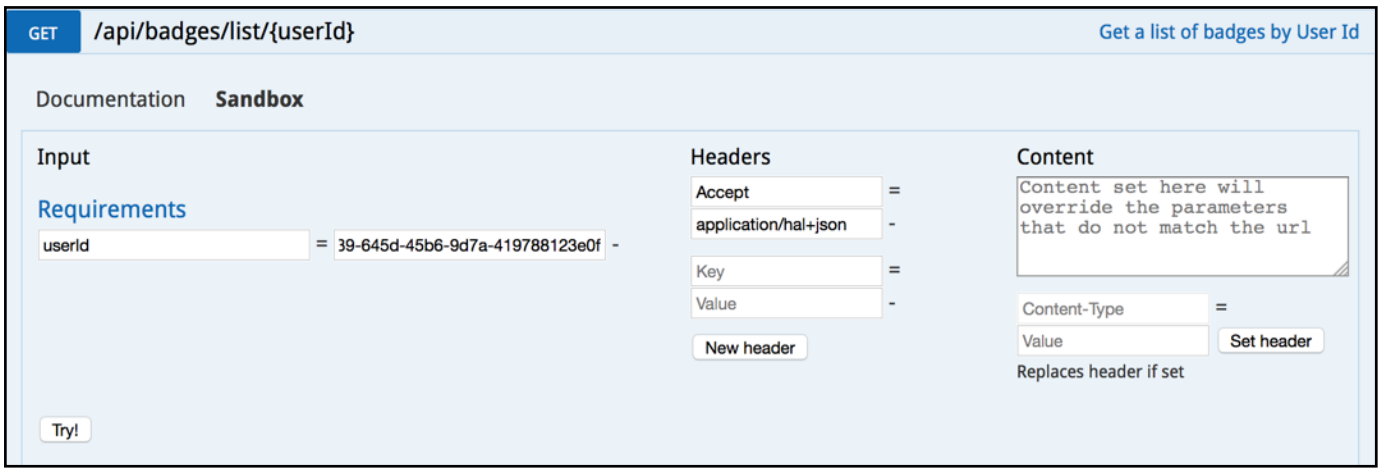


Figura 4 7.2.5 Sandbox BadgeList endpoint

A continuació es mostra el resultat que es pot observar després de realitzar l'execució de l'endpoint a través de l'entorn de sandox.

Request URL

```
GET /api/badges/list/71833939-645d-45b6-9d7a-419788123e0f
```

Response Headers [Expand]

```
200 OK
```

Response Body [Raw]


```
▿[
  ▿{
    "id": "b2b80f51-4192-47c0-a730-421840380d3a",
    "name": "Meloman",
    "description": "Li agrada molt la música",
    "is_multi_user": true,
    "_links": ▿{
      "self": ▿{
        "href": "https://badges-io-dev/api/badge/b2b80f51-4192-47c0-a730-421840380d3a/71833939-645d-45b6-9d7a-419788123e0f"
      },
      "create_badge": ▿{
        "href": "https://badges-io-dev/api/badge/create"
      },
      "delete_badge": ▿{
        "href": "https://badges-io-dev/api/badge/b2b80f51-4192-47c0-a730-421840380d3a/71833939-645d-45b6-9d7a-419788123e0f"
      },
      "update_badge": ▿{
        "href": "https://badges-io-dev/api/badge/update"
      },
      "list_badges": ▿{
        "href": "https://badges-io-dev/api/badges/list/71833939-645d-45b6-9d7a-419788123e0f"
      }
    },
    "_embedded": ▿{
      "badge:image": ▿{
        "id": "b2b80f51-4192-47c0-a730-421840380d3a",
        "name": "meloman",
        "width": 10,
        "height": 10,
        "format": "jpeg",
        "href": "https://badges-io-dev/b2b80f51-4192-47c0-a730-421840380d3a.jpeg"
      },
      "badge:owner": ▿{
        "id": "71833939-645d-45b6-9d7a-419788123e0f",
        "email": "miquel.marino@atrapalo.com",
        "username": "miquel.marino",
        "_links": ▿{
          "login": ▿{
            "href": "https://badges-io-dev/api/user/login"
          },
          "signup": ▿{
            "href": "https://badges-io-dev/api/user/signup"
          },
          "list_badges": ▿{
            "href": "https://badges-io-dev/api/badges/list/71833939-645d-45b6-9d7a-419788123e0f"
          },
          "create_badge": ▿{
            "href": "https://badges-io-dev/api/badge/create"
          }
        }
      }
    }
  }
]
```


Figura 5 7.2.5 Sandbox resultat llistat de badges per usuari

Per a poder configurar tant la documentació com l'entorn sandbox que ofereix aquest bundle, es fan servir **annotations** [64], les quals són meta-data que poden estar incrustades en el codi font. D'aquesta forma, es garanteix que la documentació es troba localitzada pròxima al codi font, facilitant el seu manteniment. A continuació es mostra les annotations en el controller de l'endpoint *BadgeList* les quals permeten generar la seva documentació així com el seu entorn de sandbox.

```
/**
 * @ApiDoc(
 *  description = "Get a list of badges by User Id",
 *  requirements={
 *    {"name"="userId", "dataType"="string", "format"="\s+", "description"=" User id", "required"="true"}
 * },
 *  statusCodes={
 *      200="Returned when successful",
 *      400="Returned when some required parameter is missing or the format its is not correct",
 *      404="Returned when user was not found",
 *      500="Returned when something when wrong"
 *  }
 * )
 * @Get("/badges/list/{userId}")
 */
public function getBadgesListAction($userId)
{
    try {
        $listBadgesCommand = $this->buildListBadgesCommandByRequest($userId);

        return $this->container->get(
            'gamification.interactor.command_handler.list_badges.list_badges_command_handler'
        )->handle($listBadgesCommand);
    } catch (\Exception $applicationException) {
        throw $this->buildBadgeHttpExceptionManager()
            ->applicationListBadgesExceptionToHttpException($applicationException);
    }
}
```

Figura 6 7.2.5 Annotations ApiDoc que permeten generar la documentació del BadgeList endpoint

8. AVALUACIÓ DEL DESACOBLAMENT DEL SISTEMA

En aquest punt del projecte, el **sistema basat en API REST ja està totalment implementat**. Ha sigut un procés progressiu el qual s'ha implementat per capes. S'ha començat pel nucli del sistema, les **capes Domain i Interactor**, les quals contenen les *entitats de domini i els caso d'ús respectivament* i els seus *tests unitaris* que garanteixen la correcta funcionalitat del sistema. El següent pas ha estat implementar la **capa Infrastructure**, escollint la *tecnologia Mysql combinada amb l'orm Doctrine* per implementar els repositoris, els quals *s'encarregaran de la persistència* de les entitats de domini i implementant els DataTransformers els quals seran els encarregats de transformar la informació de les entitats de domini en Resources per a què pugin ser exposats a capes exteriors. Finalment, s'ha implementat la **capa App**, la qual *interactuarà amb l'usuari, escollint com a tecnologia principal el framework Symfony* combinat amb bundles externs per a facilitar la implementació dels *Controllers*, els quals seran els components encarregats de definir i exposar els endpoints definits en la API Rest.

En aquest apartat **s'avalua el desacoblament de les capes de l'arquitectura del software** creant nous escenaris. D'aquesta forma es posarà a prova i s'analitzarà la capacitat d'adaptació amb casos pràctics per a poder reforçar la teoria amb dades empíriques, un dels objectius principals del projecte. Finalment, es descriurà un apartat teòric en el qual s'avaluarà el **desacoblament de l'arquitectura del sistema** amb el client i amb les màquines on s'executa.

8.1. Desacoblament arquitectura del software

Les capes que poden ser **més sensibles als canvis**, ja que depenen de la tecnologia escollida, són les capes **App i Infraestructure**. D'aquesta forma es crearan nous escenaris per avaluar l'**impacte que suposa realitzar modificacions en aquestes capes** i, **el més important, es reafirmarà el fet teòric que la modificació d'aquestes capes no implicaran modificacions ni en la capa Domain ni en la capa Interactor**.

8.1.1. Capa App

La capa App és la **responsable d'interactuar amb l'usuari**. En el sistema actual, aquesta capa consta de *Controllers* els quals defineixen els endpoints de l'API REST. El client invoca els endpoints mitjançant la comunicació utilitzant el protocol HTTP. El format de la comunicació és HAL + JSON. En aquest apartat, es plantegen nous escenaris d'interacció de l'usuari amb el sistema.

8.1.1.1. List Badges Web Controller

El model més estès en el món d'interacció amb l'usuari són les **pàgines web**. L'usuari accedeix mitjançant una URL introduïda en un navegador web a una pàgina web. Aquesta conté informació que és mostra a l'usuari en format html la qual el navegador web renderitza. L'usuari pot interaccionar amb la pàgina web mitjançant el navegador web.

En aquest apartat, s'explica **com s'ha creat una pàgina web per a llistar els badges relacionats amb un usuari.** Conté la mateixa informació que retrona l'endpoint *BadgeList* però el **format de resposta és HTML** [65] en comptes de ser HAL+JSON.

Badges List

Name	Description	IsMultiUser	Image
Melòman	Li agrada la música	Yes	
Shakespearia	Li agrada el teatre	Yes	

Figura 1 8.1.1.1 Pagina web Badges List

En la imatge anterior, es mostra el resultat de l'execució de la URL la qual mostra la pàgina web que conté el llistat d'un usuari en concret. El id de l'usuari és 1bd8d55b-0c25-48f8-a217-1eb103810d7c i la URL per mostrar el llistat de badges és https://badges-io-dev/web/badges/list/1bd8d55b-0c25-48f8-a217-1eb103810d7c.

Per a poder crear aquesta nova funcionalitat, bàsicament s'ha creat en la capa App, un **nou bundle de symfony anomenat GamificationWebBundle**.

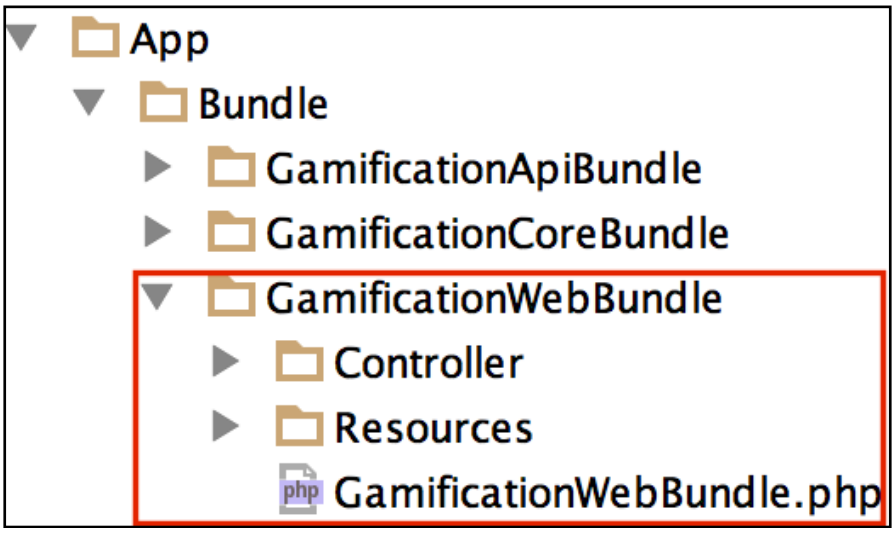


Figura 2 8.1.1.1 GamificationWebBundle

Dins el directori controller, es troba ubicat el *BadgeWebController*. També hi ha un subdirectori anomenat *WebExceptionManager* el qual conté la classe de suport *BadgeWebExceptionManager*.

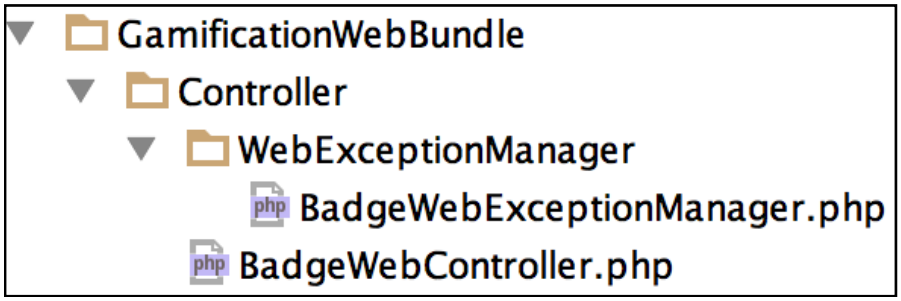


Figura 3 8.1.1.1 BadgeWebController i BadgeWebExceptionManager

```php
public function listBadgesAction($userId)
{
    try {
        $listBadgesCommand = $this->buildListBadgesCommandByRequest($userId);

        $badgesResources = $this->container->get(
            "gamification.interactor.command_handler.list_badges.list_badges_command_handler"
        )->handle($listBadgesCommand);

        return $this->renderHtmlResult($badgesResources);
    } catch (\Exception $exception) {
        return $this->renderHtmlException(
            $this->buildBadgeWebExceptionManager()
                 ->applicationListBadgesExceptionBuildErrorMessage($exception)
        );
    }
}
```

Figura 4 8.1.1.1 Mètode listBadgesAction de la classe BadgeWebController

El *BadgeWebController* conté un *action* anomenat *listBadgesAction*, el qual rep com a paràmetre un userId. En primer lloc invoca el mètode *buildListBadgesCommandByRequest* el qual crea una classe de tipus *ListBadgesCommand* passant-li com a paràmetre d'entrada l'userId. A continuació, **s'instància el ListBadgesCommandHandler mitjançant el service container i invoca el mètode handle passant-li com a paràmetre d'entrada el ListaBadgeCommand**. S'ha de notar que aquesta part de codi és exactament la mateixa que la utilitzada en el controller *BadgeApiController* en el *getBadgesListAction*. Mitjançant el service container es reaprofita i s'evita la duplicació de codi entre diferents punts del sistema. Finalment, si l'execució no retorna cap error, es crea la pàgina web amb la informació retornada pel command handler per a mostrar-la a l'usuari.

```php
/**
 * @param string $userId
 *
 * @return ListBadgesCommand
 */
private function buildListBadgesCommandByRequest($userId)
{
    return new ListBadgesCommand($userId);
}
```

Figura 5 8.1.1.1 Mètode buildListBadgesCommandByRequest

Per a poder mostrar la informació en format HTML s'utilitzen components anomenats *Templates*. Són fitxers de text els quals poden generar qualsevol tipus de fitxer en qualsevol format (HTML, XML, CSV, etc.). En el cas del framework symfony, inclou un motor de templating anomenat **Twig** [66]. Mitjançant una sintaxi oferta per Twig, el controllers de symfony poden crear fitxers HTML de forma dinàmica passant-li com a paràmetres informació de tipus array, invocant el mètode render del controller. El mètode privat *buildBadgesResourcesAsArray*, transforma una col·lecció de *BadgesResources* en una col·lecció d'arrays la qual conté la informació que es vol mostrar a l'usuari mitjançant la pàgina web.

```php
private function renderHtmlResult($badgesResources)
{
    return $this->render(
        'GamificationWebBundle::badges.list.html.twig',
        [
            'badges' => $this->buildBadgesResourcesAsArray($badgesResources)
        ]
    );
}
```

Figura 6 8.1.1.1 BadgeWebController mètode privat renderHtmlResult

```php
private function buildBadgesResourcesAsArray($badgesResources)
{
    $badgesResourcesAsArray = [];
    foreach ($badgesResources as $badgeResource) {
        $badgesResourcesAsArray[] = $this->buildBadgeResourceAsArray($badgeResource);
    }

    return $badgesResourcesAsArray;
}

/**
 * @param BadgeResource $badgeResource
 *
 * @return array
 */
private function buildBadgeResourceAsArray(BadgeResource $badgeResource)
{
    return [
        'name'          => $badgeResource->name(),
        'description'   => $badgeResource->description(),
        'isMultiUser'   => ($badgeResource->isMultiUser() == static::IS_MULTI_USER) ? 'Yes' : 'No',
        'imageHref'     => $badgeResource->imageResource()->href()
    ];
}
```

Figura 7 8.1.1.1 Mètodes privats buildBadgesResourcesAsArray i buildBadgeResourceAsArray

La informació que es mostrarà a l'usuari serà el nom del Badge, la seva descripció, si és multiuser i el link a la seva imatge física, la qual es mostrarà en la pàgina web.

Els templates de Twig, es troben ubicats al directori GamificationWebBundle/Resources/views. El template badges.list.html.twig rep la informació per part del controller i la transforma per a visualitzar-la en format HTML.

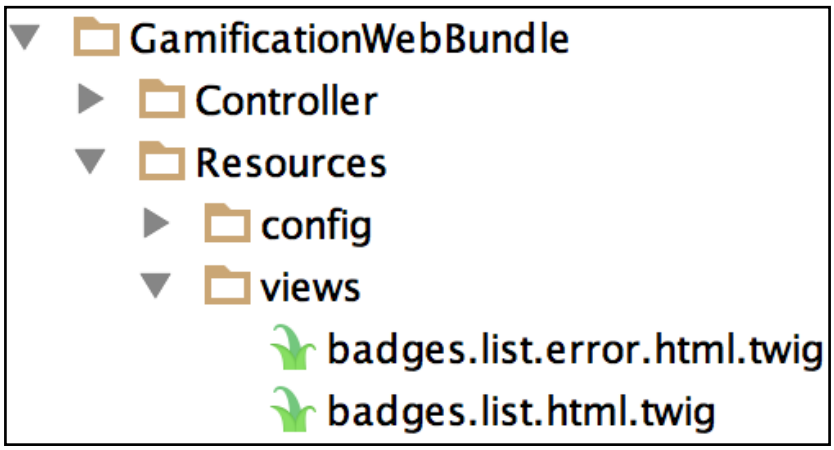


Figura 7 8.1.1.1 Directoris templates

```php
class BadgeWebExceptionManager
{
    public function applicationListBadgesExceptionBuildErrorMessage(\Exception $applicationException)
    {
        if ($applicationException instanceof InvalidListBadgesCommandException) {
            $message = "Some parameter is missing or some parameter/s format/s are wrong:" ;
        } elseif ($applicationException instanceof InvalidListBadgesCommandHandlerException) {
            $message = "Lista Badges Command Handler exception:";
        } else {
            $message = "Something went wrong :_(";
        }

        return [
            "webErrorMessage"   => $message,
            "exceptionMessage"  => $applicationException->getMessage()
        ] ;
    }
}
```

Figura 9 8.1.1.1 BadgeWebExceptionManager

En els templates també s'utilitza Bootstrap [67]. Bootstrap és el més popular HTML, CSS i JS framework per poder implementar pàgines web de tipus responsive.

En el cas de que hagi hagut algun error durant l'execució, es capturarà l'error, es processarà mitjançant la classe

BadgeWebExceptionManager i es mostrarà a través de la template badges.list.error.html.twig.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Badges IO - List Badges</title>
        <link rel="stylesheet" href="/css/bootstrap/css/bootstrap.min.css" />
        <link rel="stylesheet" href="/css/badgesList.css" />
    </head>
    <body>
        <h1 align="center">Badges List</h1>
        {% if badges | length == 0 %}
            <div class="alert alert-warning col-md-6 divCenter"  align="center">
                <strong>There is not badges for you yet! ;) </strong> <br/>
            </div>
        {% else %}
            <table class="table-bordered table-hover table-condensed badgesList" width="80%" align="center">
                <thead class="panel panel-primary">
                    <tr>
                        <th class="text-center">Name</th>
                        <th class="text-center">Description</th>
                        <th class="text-center">IsMultiUser</th>
                        <th class="text-center">Image</th>
                    </tr>
                </thead>
                {% for badge in badges %}
                    <tr>
                        <td class="text-center">
                            {{ badge.name }}
                        </td>
                        <td class="text-center">
                            {{ badge.description }}
                        </td>
                        <td class="text-center">
                            {{ badge.isMultiUser }}
                        </td>
                        <td class="text-center">
                            <img height="90" width="80" src="{{ badge.imageHref }}"/>
                        </td>
                    </tr>
                {% endfor %}
            </table>
        {% endif %}
    </body>
</html>
```

Figura 10 8.1.1.1 Template badges.list.html.twig

```
<!DOCTYPE html>
<html>
    <head>
        <title>Badges IO - List Badges</title>
        <link rel="stylesheet" href="/css/bootstrap/css/bootstrap.min.css" />
        <link rel="stylesheet" href="/css/badgesList.css" />
    </head>
    <body>
        <h1 align="center">Badges List</h1>

            <div class="alert alert-danger col-md-6 divCenter" align="center">
                <strong>Ooops! It has been an error! </strong> <br/>
                <strong> {{ webErrorMessage }} </strong> <br/>
                Details: {{ exceptionMessage }}
            </div>

    </body>
</html>
```

Figura 11 8.1.1.1 Template badges.list.error.html.twig

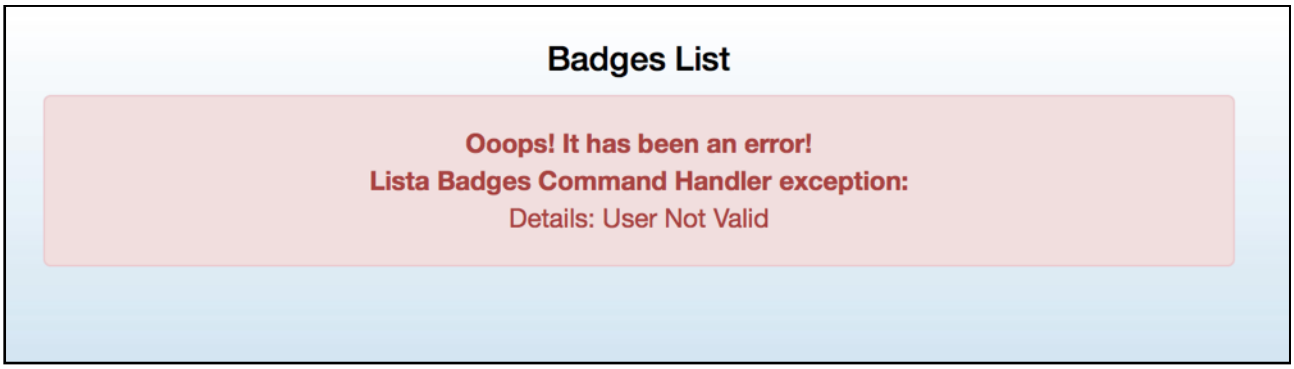


Figura 12 8.1.1.1 Pàgina web error usuari no es vàlid

8.1.1.2. Anàlisi impacte integració web controller

En aquest apartat es realitza l'anàlisi de l'impacte de la integració web controller. Es mesurarà l'impacte en nombre de fitxers, funcions, línies de codi creades i ubicació del nou codi en l'arquitectura software que ens donarà una idea del grau de dispersió de codi.

Per a poder crear la funcionalitat de la pàgina web s'han creat un total de **6 fitxers**: BadgeWebExceptionManager.php, BadgeWebController, routing.yml, badges.list.error.html.twig, badges.list.html.twig i GamificationWebBundle.

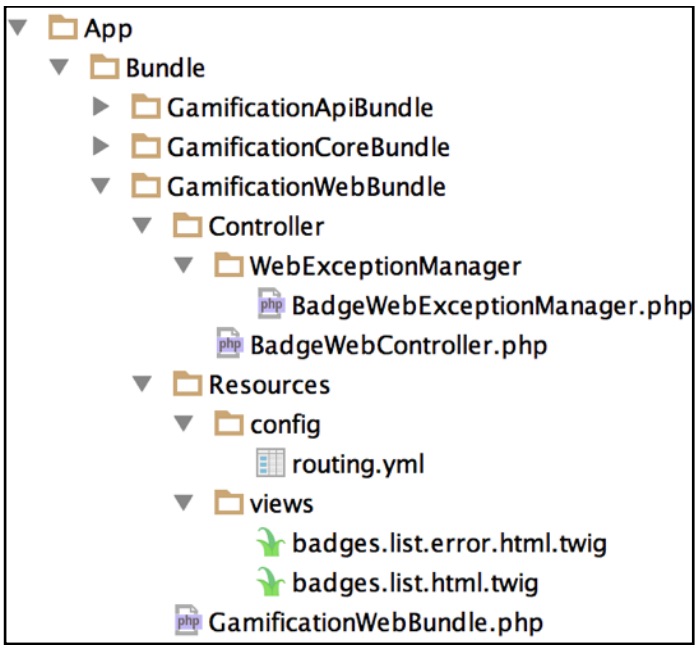


Figura 1 8.1.1.3 Directoris GamificationWeb

Els fitxers routing.yml i GamificationWebBundle són fitxers els quals defineixen la ruta per accedir mitjançant una URL al controller i la definició de la configuració del bundle respectivament.

Els fitxers consten de **8 mètodes** en total:

- 1 mètode de la classe BadgeWebExceptionManager
- 7 mètodes de la classe BadgeWebController

Els components que implementen aquesta funcionalitat, consten de **212 línies en total**:

- 26 línies del fitxer BadgeWebExceptionManager.php
- 111 línies del fitxer BadgeWebController.php
- 4 línies del fitxer routing.yml
- 18 línies del fitxer badges.list.error.html.twig
- 43 línies del fitxer badge.list.html.twig
- 10 línies del fitxer GamificationWebBundle.php

S'ha de notar que tots els **fitxers que defineixen la funcionalitat, estan encapsulats en el directori GamificationWebBundle el qual s'ha creat nou dins la capa App. No s'ha modificat ni creat cap component en la resta de capes de l'arquitectura**.

8.1.1.3. List Badges Command Console

El framework symfony, ofereix la possibilitat de poder crear línies de comandes per consola [68]. En aquest apartat, es crearà una línia de comanda per a llistar els badges d'un usuari per terminal.

Tot i no ser estrictament un bundle per si mateix, s'han ubicat els components que formen el command dins del directori App/Bundle. S'ha pres aquesta decisió perquè no es creu que tingui prou entitat el command per ser una entitat externa al bundles de la capa App i per a facilitar la seva identificació i la cerca dels components.

Figura 1 8.1.1.3 Directoris GamificationCommand

Dins del directori GammificationCommand, es troba la classe *ListBadgesCommand*. Dins el directori CommandExceptionManager es troba ubicada la classe *BadgeCommandExceptionManager*.

La classe LitsBadgesCommand té un mètode anomenat execute que rep un canal d'entrada i un canal de sortida com a paràmetres. El primer conté la informació que li ha passat a l'usuari en l'execució de la comanda en un terminal i el segon conté el canal per a mostrar el resultat a l'usuari.

```php
protected function execute(InputInterface $input, OutputInterface $output)
{
    try {
        $listBadgesCommand = $this->buildListBadgesCommandByRequest($input->getArgument('userId'));

        $badgesResources = $this->getContainer()->get(
            "gamification.interactor.command_handler.list_badges.list_badges_command_handler"
        )->handle($listBadgesCommand);

        $this->showCommandResult($output, $input->getArgument('userId'), $badgesResources);
    } catch (\Exception $exception) {
        $this->buildCommandExceptionManager()
             ->applicationListBadgesExceptionShowCommandError($output, $exception);
    }
}
```

Figura 2 8.1.1.3 ListBadgesCommand mètode execute

Aquest mètode, invoca el mètode privat *buildListBadgesCommandByRequest*, el qual crea un objecte de tipus ListBadgesCommand de la capa Interactor.

```php
private function buildListBadgesCommandByRequest($userId)
{
    return new ListBadgesCommand($userId);
}
```

Figura 3 8.1.1.3 ListBadgesCommand mètode buildListBadgesCommandByRequest

A continuació, instància mitjançant el service container la classe *ListBadgesCommandHandler* per a executar tot seguit el mètode handle passant-li com a paràmetre el *ListBadgesCommand*.

Finalment i si l'execució no ha tingut cap error, es mostra el resultat a l'usuari mitjançant el mètode privat *showCommandResult*.

S'ha de notar que tant en el cas de l'endpoint de l'API Rest, com el del web controller que llista els badges com el del command, **es comparteix el codi de la creació de l'objecte ListBadgesCommandHandler mitjançant el service container. D'aquesta forma, es reaprofita una vegada més el codi i s'evita la seva duplicació**.

En el cas de què hi hagi hagut algun error durant l'execució, el mètode execute capturarà l'excepció i mostrarà un missatge d'error mitjançant el mètode públic

applicationListBadgesExceptionShowCommandError de la classe *BadgeCommandExceptionManager*.

Finalment en aquest apartat es mostra el codi de la classe *BadgeCommandExceptionManager*, una execució del command sense error i una execució del command amb error d'execució.

```php
public function applicationListBadgesExceptionShowCommandError(
    OutputInterface $output,
    \Exception $applicationException
) {
    if ($applicationException instanceof InvalidListBadgesCommandException) {
        $message = "Some parameter is missing or some parameter/s format/s are wrong:" ;
        $errorMessage = $this->applyFormatToMessage($message) . "|\n";
    } elseif ($applicationException instanceof InvalidListBadgesCommandHandlerException) {
        $message = "Lista Badges Command Handler exception:";
        $errorMessage = $this->applyFormatToMessage($message) ."|\n";
    } else {
        $message = "Something went wrong :_(";
        $errorMessage = $this->applyFormatToMessage($message) ."|\n";

    }

    $errorMessage .= $this->applyFormatToMessage($applicationException->getMessage()) . "|";
    $this->showCommandErrorMessage($output, $errorMessage);
}
```

Figura 4 8.1.1.3 BadgeCommandExceptionManager mètode públic per mostrar l'error a l'usuari

Per executar la command s'ha d'obrir un terminal, situar-se al directori arrel del projecte i executar **app/console gamification:list-badges 1bd8d55b-0c25-48f8-a217-1eb103810d7c** on 1bd8d55b-0c25-48f8-a217-1eb103810d7c és el id del usuari.

```
|------------------------------------------------------------------------------|
|        BADGES LIST BY USER ID 1bd8d55b-0c25-48f8-a217-1eb103810d7c            |
|------------------------------------------------------------------------------|
|name:        Melòman                                                          |
|description: Li agrada la música                                             |
|isMultiUser: Yes                                                              |
|image href:  https://badges-io-dev/38d5ff36-7bd8-4e1e-b586-7296874f031c.jpeg  |
|------------------------------------------------------------------------------|
|name:        Shakespearia                                                     |
|description: Li agrada el teatre                                             |
|isMultiUser: Yes                                                              |
|image href:  https://badges-io-dev/9723b9c8-d961-4c1c-98ae-3cb53a563044.png   |
|------------------------------------------------------------------------------|
```

Figura 5 8.1.1.3 Execució ListBadgesCommand sense error

```
|-------------------------------------------------------------|
|     LIST BADGES COMMAND ERROR                               |
|     Lista Badges Command Handler exception:                 |
|     User Not Valid                                          |
|-------------------------------------------------------------|
```

Figura 6 8.1.1.3 Execució ListBadgesCommand amb error

8.1.1.4. Anàlisis impacte integració command console

En aquest apartat es realitza l'anàlisi de l'impacte de la integració del command. Es mesurarà l'impacte en nombre de fitxers, funcions, línies de codi creades i ubicació del nou codi en l'arquitectura software que ens donarà una idea del grau de dispersió de codi.

Per a poder crear la funcionalitat de la pàgina web s'han creat un total de **2 fitxers**: BadgeCommandExceptionManager.php i el ListaBadgesCommand.php.

Figura 7 8.1.1.3 GamificationCommand directoris

Els fitxers consten de **13 mètodes** en total:

- 4 mètodes de la classe BadgeCommandExceptionManager
- 9 mètodes de la classe ListBadgesCommand

Els components que implementen aquesta funcionalitat, consten de **226 línies en total**:

- 82 línies del fitxer BadgeCommandExceptionManager.php
- 144 línies del fitxer ListBadgesCommand.php

S'ha de notar que tots els **fitxers que defineixen la funcionalitat, estan encapsulats en el directori GamificationCommand el qual s'ha creat nou dins la capa App. No s'ha modificat ni creat cap component en la resta de capes de l'arquitectura**.

8.1.2. Capa Infrastructure

En aquest apartat, es plantejarà substituir la tecnologia que implementa la capa Infrastructure. Aquest fet suposà modificar com s'obté, es modifiquen i es persisteixen les dades en el sistema. Finalment, s'analitzarà l'impacte en el sistema que suposa la substitució de la tecnología.

8.1.2.1. Redis Repositori

Tal com s'ha comentat en l'apartat on es descrivia la capa Infrastructure, la tecnologia escollida per implementar la persistència de dades del sistema, és a dir per implementar els repositoris, és la base de dades de tipus *Mysql*.

En aquest apartat és substituirà la tecnologia de tipus *Mysql* per la tecnologia de persistència de tipus **Redis** [69]. *Redis* és un emmagatzematge d'estructures de dades en memòria el qual s'utilitza com a base de dades o cache de dades.

Mitjançant un client de *Redis*, mitjançant un terminal es pot connectar per a observar com redis emmagatzema les dades. *Redis* emmagatzema la informació mitjançant el sistema clau-valor. És a dir, mitjançant una clau de forma única s'obté la informació emmagatzemada. A continuació es mostra com la informació es guarda amb aquest tipus de tecnologia mitjançant un terminal de consola.

```
MAC-0215-2:badges-io miquel.marino$ redis-cli -h badges-io-dev
badges-io-dev:6379> keys *
(empty list or set)
badges-io-dev:6379>
```

Figura 1 8.1.2.1 Client redis. Estat inicial no hi ha informació emmagatzemada

```
badges-io-dev:6379> keys BADGE_*
1) "BADGE_af4eee26-8ea6-444d-b255-1d5a7b450617"
2) "BADGE_2c35ded0-031c-4391-8919-dfd8d419980c_621c3c4b-83a6-4242-b149-79bebce7e915_YES"
3) "BADGE_af4eee26-8ea6-444d-b255-1d5a7b450617_621c3c4b-83a6-4242-b149-79bebce7e915_YES"
4) "BADGE_2c35ded0-031c-4391-8919-dfd8d419980c"
```

Figura 2 8.1.2.1 Llistat de claus de tipus Badge un cop creats dos badges

```
badges-io-dev:6379> get BADGE_af4eee26-8ea6-444d-b255-1d5a7b450617
"O:25:\"Domain\\Entity\\Badge\\Badge\":6:{s:29:\"\x00Domain\\Entity\\Badge\\Badge\x00id\";s:36:\"af4eee26-8
ea6-444d-b255-1d5a7b450617\";s:31:\"\x00Domain\\Entity\\Badge\\Badge\x00name\";s:8:\"Mel\xc3\xb2man\";s:38:
\"\x00Domain\\Entity\\Badge\\Badge\x00description\";s:20:\"Li agrada la m\xc3\xbasica\";s:38:\"\x00Domain\\
Entity\\Badge\\Badge\x00isMultiUser\";b:1;s:31:\"\x00Domain\\Entity\\Badge\\Badge\x00user\";O:23:\"Domain\\
Entity\\User\\User\":4:{s:27:\"\x00Domain\\Entity\\User\\User\x00id\";s:36:\"621c3c4b-83a6-4242-b149-79bebc
e7e915\";s:30:\"\x00Domain\\Entity\\User\\User\x00email\";s:26:\"miquel.marino@atrapalo.com\";s:33:\"\x00Do
main\\Entity\\User\\User\x00userName\";s:13:\"miquel.marino\";s:33:\"\x00Domain\\Entity\\User\\User\x00pass
Word\";s:32:\"1224906adf7690a0e38d807547227409\";}s:32:\"\x00Domain\\Entity\\Badge\\Badge\x00image\";O:25:\
"Domain\\Entity\\Image\\Image\":5:{s:29:\"\x00Domain\\Entity\\Image\\Image\x00id\";s:36:\"af4eee26-8ea6-444
d-b255-1d5a7b450617\";s:31:\"\x00Domain\\Entity\\Image\\Image\x00name\";s:8:\"Mel\xc3\xb2man\";s:32:\"\x00D
omain\\Entity\\Image\\Image\x00width\";i:20;s:33:\"\x00Domain\\Entity\\Image\\Image\x00height\";i:20;s:33:\
"\x00Domain\\Entity\\Image\\Image\x00format\";s:4:\"jpeg\";}}"
```

Figura 3 8.1.2.1 Contingut emmagatzemat d'una clau de tipus badge

Per implementar els repositoris de tipus *Redis*, s'ha creat un directori dins de la ruta Infrastructure/Persistence anomenat *Redis*. En el interior d'aquest directori, hi ha la ruta Domain/Entity. En aquesta ruta hi ha el directori amb el nom de cada entitat. Seguint amb el format establert, el nom dels repositoris tenen com a prefix el nom de la tecnologia que els implementa, en aquest cas *Redis*.

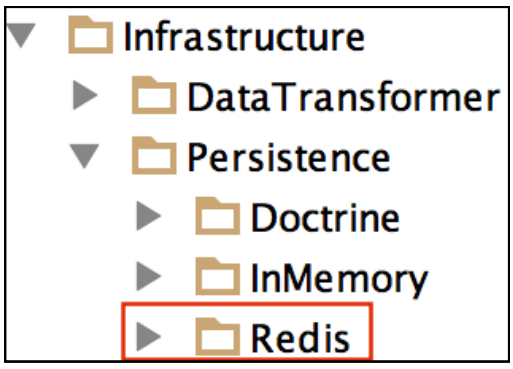


Figura 4 8.1.2.1 Redis directori

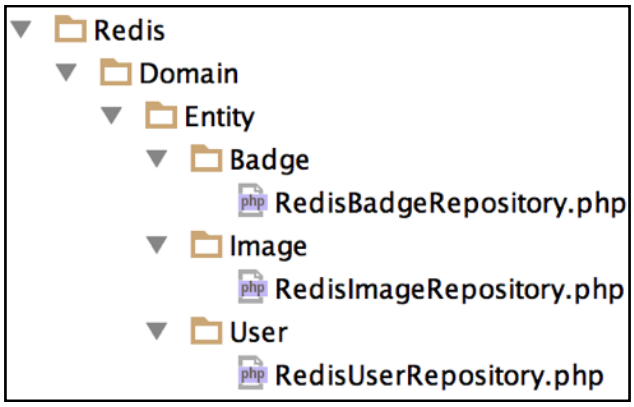


Figura 5 8.1.2.1 Redis repositoris

A continuació, s'il·lustra amb l'exemple del *RedisBadgeRepository* la implementació d'un repositori de tipus redis. Aquest repositori implementara els mètodes de la interfície *BadgeRespository*.

```php
interface BadgeRepository
{
    /**
     * @param Badge $badge
     */
    public function persist(Badge $badge);

    /**
     * @param string $id
     *
     * @return Badge | null
     */
    public function find($id);

    /**
     * @param Badge $badge
     */
    public function remove(Badge $badge);

    /**
     * @param User $user
     *
     * @return Badge[]
     */
    public function findByUser(User $user);

    /**
     * @return Badge[]
     */
    public function findMultiUser();
}
```

Figura 6 8.1.2.1 BadgeRepository domain

Al mètode constructor de la classe *RedisBadgeRepository*, se li injecta un client de *Redis* implementat en php per a poder gestionar aquesta tecnologia. El client utilitzat en aquest cas és Predis [70].

```php
class RedisBadgeRepository implements BadgeRepository
{
    const REDIS_KEY_BADGE_PREFIX = "BADGE_";

    /**
     * @var Client
     */
    private $client;

    public function __construct(Client $client)
    {
        $this->client = $client;
    }
```

Figura 6 8.1.2.1 RedisBadgeRepository mètode constructor

A continuació, es mostra com s'implementa el mètode *find*. Mitjançant la connexió del client, s'executa una comanda *get* mitjançant una clau d'emmagatzematge amb el prefix BADGE_ i com a sufix l'id rebut per paràmetre. *Redis* emmagatzema com a valor l'entitat serialitzada. Amb la versió del llenguatge Php 7, el mètode *unserialize* retorna una entitat de tipus *Badge*.

```php
public function find($id)
{
    return $this->retrieveBadge(static::REDIS_KEY_BADGE_PREFIX . $id);
}
```

Figura 7 8.1.2.1 RedisBadgeRepository mètode find

```php
private function retrieveBadge($key)
{
    $badgeInfo = $this->client->get($key);

    if (!$badgeInfo) {
        return null;
    }

    return unserialize($badgeInfo);
}
```

Figura 8 8.1.2.1 RedisBadgeRepository retrieveBadge

D'aquesta forma, s'implementaran tots els repositoris amb la nova tecnologia, implementant els mètodes declarats en les interfícies de la capa Domain.

Per a poder injectar els nous repositoris als *CommandHandlers*, **s'ha de modificar com els controllers de la capa App injecten els repositoris**. Més concretament, s'haurà de **modificar la configuració del service container**, on s'instancien els *CommandHandlers* com a serveis i on es realitza la injecció de dependències.

En primer lloc, es crearà el fitxer de configuració del service container el qual registra els *RedisRepositories* com a serveis. Aquest fitxer s'anomena redis-repositories.xml i es troba ubicat a la ruta App/Bundle/GamificationCoreBundle/Resources/config/container/infrastructure/persistence/repositoris/redis-repositoris.

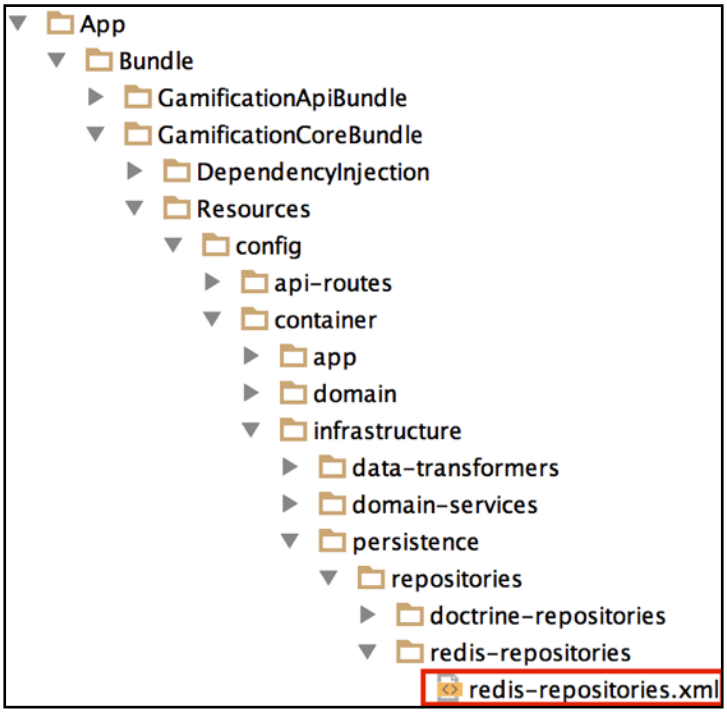


Figura 9 8.1.2.1 Localització redis-repositories.xml

La configuració dels repositoris de redis consisteix a instanciar com a servei el client Predis passant-li com a paràmetre d'entrada la configuració per realitzar la connexió. La configuració conté la màquina i el port al qual s'ha de connectar.

```xml
<?xml version="1.0" ?>

<container xmlns="http://symfony.com/schema/dic/services"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://symfony.com/schema/dic/services http://symfony.com/schema/dic/services/services-1.0.xsd">

    <services>
        <service id="gamification.predis.client"
                 class="Predis\Client">
            <argument>%redis_client_conf%</argument>
        </service>

        <service id="gamification.infrastructure.persistence.redis.domain.entity.badge.redis_badge_repository"
                 class="Infrastructure\Persistence\Redis\Domain\Entity\Badge\RedisBadgeRepository">
            <argument type="service" id="gamification.predis.client" />
        </service>

        <service id="gamification.infrastructure.persistence.redis.domain.entity.image.redis_image_repository"
                 class="Infrastructure\Persistence\Redis\Domain\Entity\Image\RedisImageRepository">
            <argument type="service" id="gamification.predis.client" />
        </service>

        <service id="gamification.infrastructure.persistence.redis.domain.entity.user.redis_user_repository"
                 class="Infrastructure\Persistence\Redis\Domain\Entity\User\RedisUserRepository">
            <argument type="service" id="gamification.predis.client" />
        </service>
    </services>

</container>
```

Figura 10 8.1.2.1 Service Container redis-repositories.xml

Cada repositori que es configura com a servei, rep com a paràmetre d'entrada el client al qual es referència mitjançant el seu identificador de servei.

Finalment, es modifica el fitxer repositories.xml del directori domain, el qual conté els alies dels repositoris de la capa Infrastructure i serveixen com a capa d'abstracció. D'aquesta forma, **en la configuració del service container, s'injecten als CommandHandlers els alies els quals tenen com a nom les interfícies de la capa Domain**. Aquesta configuració fa que el codi sigui més reaprofitable i facilita el manteniment.

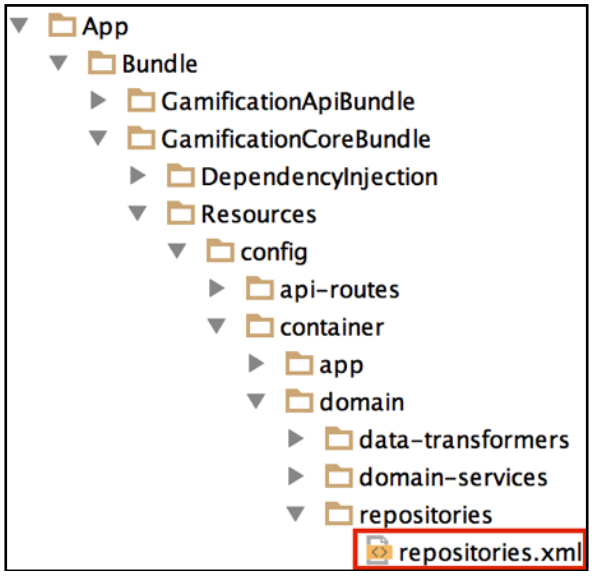


Figura 11 8.1.2.1 Domain repositories

```xml
<?xml version="1.0" ?>

<container xmlns="http://symfony.com/schema/dic/services"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://symfony.com/schema/dic/services http://symfony.com/schema/dic/services/services-1.0.xsd">

    <services>
        <service id="gamification.domain.entity.badge.badge_repository"
                 alias="gamification.infrastructure.persistence.redis.domain.entity.badge.redis_badge_repository" />

        <service id="gamification.domain.entity.image.image_repository"
                 alias="gamification.infrastructure.persistence.redis.domain.entity.image.redis_image_repository" />

        <service id="gamification.domain.entity.user.user_repository"
                 alias="gamification.infrastructure.persistence.redis.domain.entity.user.redis_user_repository" />

    </services>

</container>
```

Figura 12 8.1.2.1 Domain repositories.xml

D'aquesta forma, no serà necessari modificar la injecció de dependències del fitxer de configuració dels *CommandHandlers*, ja que s' injecten els alies del fitxer repositories.xml de la configuració domain. Aquests reben com a paràmetre d'entrada els alies que s'han creat. Només és necessari modificar a quins repositoris fan referència els alies, és a dir, substituir la referència dels doctrine repositoris pels de redis repositoris.

Com es pot observar en la configuració command-handlers.xml, li passa com a paràmetre d'entrada el alies del BadgeRepository i l'UserRepository. No instància directament els repositoris de la capa d'Infrastructure.

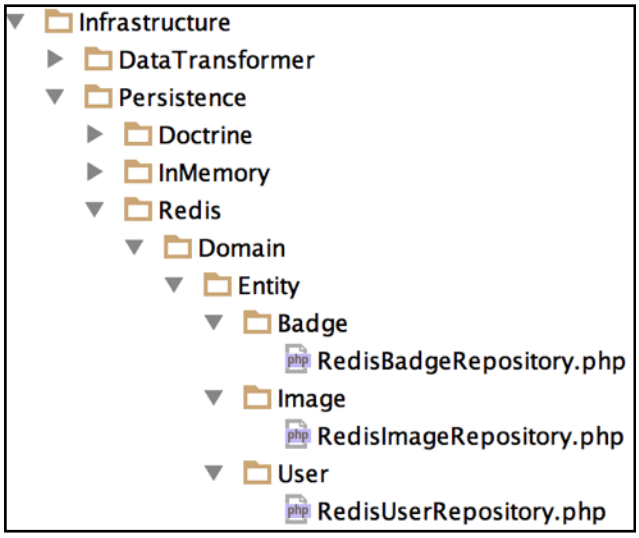
```
<service id="gamification.interactor.command_handler.list_badges.list_badges_command_handler"
         class="Interactor\CommandHandler\ListBadges\ListBadgesCommandHandler">
    <argument type="service" id="gamification.domain.entity.badge.badge_repository" />
    <argument type="service" id="gamification.domain.entity.user.user_repository" />
    <argument type="service" id="gamification.domain.entity.badge.badge_collection_data_transformer" />
</service>
```

Figura 13 8.1.2.1 Configuració ListBadgesCommandHandler command-handlers.xml

S'ha de notar que **no és necessari modificar el codi dels CommandHandlers**, ja que tots els repositoris de les diferents tecnologies (Doctrine, InMemory o Redis), implementen les interfícies **dels Repositoris de la capa Domain els quals són els que finalment s'injecten als CommandHandlers**. Pel que implementen els mètodes els quals són utilitzats pels *CommandHandlers* per executar el codi dels casos d'ús.

8.1.2.2. Anàlisis impacte integració redis repositoris

En aquest apartat es realitza l'anàlisi de l'impacte de la integració del command. Es mesurarà l'impacte en nombre de fitxers, funcions, línies de codi creades i ubicació del nou codi en l'arquitectura software que ens donarà una idea del grau de dispersió de codi.



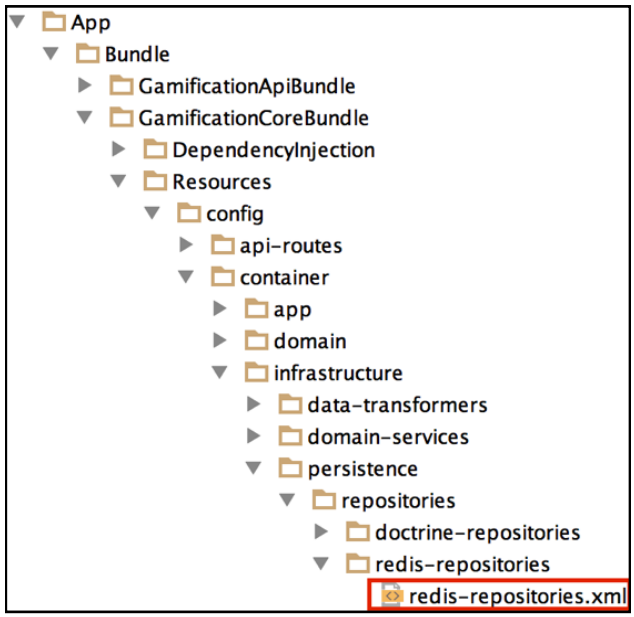


Figura 1-2 8.1.2.2 Fitxers RedisRepositoris capa Infrastructure i redis-repositories.xml capa App

Per a poder crear la funcionalitat de la pàgina web s'han creat un total de **4 fitxers**: RedisBadgeRepository.php, RedisImageRepository.php, RedisUserRepository.php i redis-repositories.xml. Els 3 primers estan ubicats en la capa App i l'últim està ubicat a la capa App.

Els fitxers consten de **25 mètodes** en total:

- 9 mètodes de la classe RedisBadgeRepository
- 6 mètodes de la classe RedisImageRepository
- 10 mètodes de la classe RedisUserRepository

Els components que implementen aquesta funcionalitat, consten de **361 línies en total**:

- 143 línies del fitxer RedisBadgeRepository.php
- 78 línies del fitxer RedisImageRepository.php
- 110 línies del fitxer RedisUserRepository.php
- 30 línies del fitxer redis-repositories.xml

Adicionalment, s'han **modificat 3 línies de codi** del fitxer de configuració del service container repositories.xml.

Per a realitzar la integració dels nous repositoris, ha sigut necessari **crear nous components en la capa Infrastructure** els quals són els nous repositoris implementats amb la nova tecnologia i **modificar en la capa App com els components (controllers i command) realitzen la injecció dels repositoris** als CommandHandler mitjançant la configuració del service container. **No s'ha modificat ni la capa Interactor ni la capa Domain**. Tots els **nous components que s'han creat s'ubiquen dins la capa Infrastructure**.

8.1.3. Conclusions

En la següent taula resum, es recull les dades analitzades en els tres escenaris.

	WEB CONTROLLER	CONSOLE COMMAND	REDIS REPOSITORIS
FITXERS	6	2	4
MÈTODES	8	13	25
LÌNIES	212	226	361

Figura 1 8.1.3 Dades impacte integració dels diferents escenaris

Amb les dades obtingudes i una vegada realitzat l'anàlisi detallat de l'impacte que suposa cada nou escenari, es poden extreure les següents conclusions:

- **L'escenari de canvi de tecnologia d'infraestructura té l'impacte més gran d'integració**. És a dir, l'escenari de modificar la tecnologia d'implementació de la capa Infrastructure, suposa crear el doble de mètodes i un factor de creixement de 1,5 línies de codi (3 / 2) més comparat amb la creació del console command de la capa App.
- **El sistema presenta un factor d'escalabilitat raonable i assumible**. Aquest fet significa que si es vol crear un nou component per la capa App o modificar la implementació de la capa Infrastructure per una altra tecnologia, les dades obtingudes realitzant els nous canvis haurien de ser molt semblants a les que s'han mesurat en els diferents escenaris plantejats, les quals demostren la viabilitat i la capacitat d'adaptació del sistema.
- **L'arquitectura software està desacoblada**. S'ha pogut observar que en els escenaris on s'introdueixen nous elements en la capa *App* no ha estat necessari modificar la resta de capes. Pel que fa al canvi en la capa *Infrastructure* ha estat necessari modificar quins repositoris s'injectaven als *CommandHandlers* per part dels components de la capa *App* però aquest fet ja és una de les responsabilitats assumibles que tenen aquests components. També s'ha de tenir en compte que els nous

components es troben ben identificats i ubicats en els directoris de l'arquitectura i són components desacoblats dins de les mateixes capes de la resta els quals ja existien prèviament. La dispersió dels nous components creats i ubicats en l'arquitectura és baixa, presentant l'arquitectura software una gran cohesió en vers dels canvis.

- **En cap escenari s'ha modificat o creat nous components ni en la capa Domain ni en la capa Interactor**. Aquesta potser és la conclusió més important i un gran valor que ens aporta l'arquitectura del software dissenyada pel sistema actual. Cap escenari, ni per modificar com es presenten les dades a l'usuari o com aquest interactua amb el sistema (web controller, command console) ni per substituir la tecnologia utilitzada per persistir les dades (redis repositoris), ha creat la necessitat de modificar els elements de les capes Domain i Interactor. Gràcies a l'abstracció dels repositoris com a interfícies de la capa Domain, només és necessari que els repositoris de la capa Infrastructure implementin els mètodes definits en les interfícies per evitar modificacions en la capa Interactor, més concretament en els components CommandHandlers. Tal com s'ha comentat, les capes Domain i Interactor representen el nucli del sistema. Es redueix la modificació del codi dels components d'aquestes capes tan sols en les situacions en la que es modifiqui la lògica de negoci, evitant possibles errors greus del funcionament del sistema i garanteix el seu manteniment. Ja que, substituir la tecnologia per persistir les dades per Redis en comptes de Mysql no hauria d'influir en el flux del procés del component de crear un badge. El mateix fet equival quan es modifica l'endpoint de l'api rest list badges per una pàgina web que llista els badges d'un usuari.
- **Totes les conclusions anteriors, garanteixen el baix cost de manteniment i escalabilitat del sistema**.

8.2. Desacoblament arquitectura del sistema

En l'apartat anterior, s'ha realitzat un anàlisi pràctic demostrant el desacoblament de l'arquitectura software. En aquest apartat, es realitzarà un anàlisi teòric avaluant el desacoblament de l'arquitectura del sistema. És a dir, com la decisió de **crear una API Rest i en definitiva crear un sistema SaS, aporta desacoblament també amb les màquines on el software s'executa i amb els clients que utilitzen el sistema**.

8.2.1. Infraestructura del sistema

En aquest apartat es comenta com el sistema està desacoblat de les màquines i l'entorn físic on s'executa.

Actualment, el sistema es troba allotjat on està la resta d'aplicacions de l'empresa i l'allotjament és local. No obstant això, el fet de què **la resta del sistema interactua amb la nova eina de forma desacoblada amb la nova eina a través dels endpoints i el fet que la persistència també està desacoblada de la resta, la nova eina es pot reubicar en altres màquines amb un baix cost de portabilitat**. Aquest fet suposa avantatges tan interessants com poden ser migrar la nova eina a altres màquines, fins i tot, portar-lo a un entorn cloud [71][72] amb totes les millores que suposaria.

El baix cost en la portabilitat i tenir desacoblat el sistema de la infraestructura, pot permetre en un futur pròxim plantejar escenaris com poden ser el **balancejat de la infraestructura i poder aplicar paradigmes de sistemes distribuïts**. D'aquesta forma, a mesura que la càrrega del sistema augmenti pel fet de l'augment de clients que utilitzen la nova eina, el desacoblament de l'arquitectura del sistema facilita la mesura de quina infraestructura és necessària per suportar una certa càrrega i les característiques que aquesta infraestructura ha de tenir per a tenir un temps de resposta raonable. **Aquesta característica permet la possibilitat de configurar un sistema d'alta disponibilitat** [73].

8.2.2. Client API

L'usuari el qual utilitza l'eina tan sols té les necessitats de conèixer com invocar l'API Rest per a realitzar una funcionalitat concreta i el tipus de resposta (format i informació) que li retorna la API Rest. Gràcies al fet de què l'API Rest és de nivell 3, ni tan sols té la necessitat prèvia de conèixer totes les funcionalitats que ofereix l'API, ja que aquesta anirà incluoent en les successives respostes endpoints relacionats per a què l'usuari pugui navegar per l'API Rest.

D'aquesta forma, **es facilita el desenvolupament de clients els quals consulten l'API Rest en diferents tecnologies**. **Un client de l'API Rest és una capa d'abstracció la qual facilita la invocació dels endpoints**, la informació dels paràmetres, el verb HTTP que s'utilitza i la inclusió de l'API key per a què sigui més usable per part de l'usuari. **Una bona pràctica és que l'equip de programadors que mantenen l'API Rest també ofereixin clients de l'API** en diferents tecnologies i que estiguin mantingudes pel mateix equip. En el cas, per exemple, de què es modifiqui una URL d'un endpoint o se'n crei un de nou, actualitzant només el client el canvi seria transparent a l'usuari.

Per un altra banda, els escenaris exposats durant l'avaluació de l'impacte de la creació de nous components de la capa App, el web controller i el console command podrien haver invocat l'endpoint BadgeList en lloc d'invocar el command handler, afegint un nou nivell de desacoblament. **En aquest cas, el codi dels *CommandHandlers* i del web controller i del command no tindrien la necessitat de residir en la mateixa màquina**. Això significa, que un usuari extern pot crear una pàgina web o una comanda de consola invocant els endpoints de l'API. **Aquest fet demostra que l'usuari de l'API i l'API en si estan desacoblats**.

Es poden crear diferents clients API per a diferents tecnologies (php, java, javascript, python, mobile, etc.) per incloure el màxim tipus d'usuari possible. **Poder crear clients amb diferents tipus de tecnologies les quals poden ser diferents de la tecnologia en la qual està implementada els endpoints de l'API Rest (per exemple el llenguatge de programació) demostra el fet de la independència tecnològica entre l'usuari i la nova eina**, un dels objectius principals dels sistemes SaaS.

9. CONCLUSIONS FINALS

El projecte ha consistit a crear **una eina open source la qual permeti gestionar perfils d'usuari** per a assolir l'**objectiu d'incrementar el ràtio de conversió d'usuaris que acaben realitzant una compra** en el portal web Atrapalo.

D'aquesta forma i per assolir l'objectiu plantejat a l'inici del projecte, s'ha creat una **eina basada en el concepte de Gamification** la qual permet gestionar badges i assignar-los als usuaris per incrementar la seva fidelització al sistema. La **nova eina està desacoblada a nivell software del portal web Atrapalo** encara que actualment s'executa en el mateix hardware on es troba la resta del portal web.

La creació de l'eina s'ha basat en un **sistema de microservei mitjançant l'arquitectura API Rest** creant un sistema de tipus SaaS. Els components software que conformen el sistema han estat implementats mitjançant **l'arquitectura hexagonal, una arquitectura de capes que garanteix el desacoplament dels components entre si**.

El disseny de l'eina s'ha realitzat de forma progressiva per capes, començant pel **nucli el qual està conformat per la capa Domain i la capa Interactor**. El nucli del sistema està **totalment testejat** per tests unitaris garantint la robustesa del codi.

A continuació s'han implementat les capes més externes i més **dependents de la tecnologia**, la capa **Infrastructure i la capa App** les quals estan implementades basant-se bàsicament amb la tecnologia base de dades **Mysql** i el **framework Symfony** respectivament. S'ha de notar que fins a la fase d'implementació d'aquestes capes no s'ha escollit quines tecnologies serien les més adequades per implementar el sistema.

S'han i**mplementat els endpoints de les API Rest en la capa App així com una documentació online** i dinàmica oferta a l'usuari per a conèixer les funcionalitats que ofereix l'Api.

Finalment en els últims apartats, **s'ha demostrat mitjançant diferents escenaris teòrics i pràctics del desacoblament que ofereix el sistema** .

L'objectiu tecnològic ha estat assolit. El sistema s'està executant en l'entorn de producció de la web amb molts bons resultats a nivell quantitativament i qualitativament parlant.

10. PRESSUPOST

En el pressupost es té en compte les hores necessàries per a què el programador que dissenyi i implementi l'eina. Són un total de 600 hores d'un programador senior en PHP.

Pel que fa a les tecnologies escollides, tant Mysql, Symfony i Nginx són de tipus open source pel que no afegeixen cost a pressupost.

Pel que fa als servidors, com s'ha optat per l'opció de mantenir l'eina en els servidors que ja existeixen pel portal web, no afegeixen cost al pressupost. En un futur és possible que aquest cost sigui el més sensible a què augmenti en el cas que es decideixi invertir en servidors dedicats per la nova eina.

El pressupost total són les hores del programador multiplicat pel seu import per hora que són uns 60 €. El resultat són **36.000 €** de pressupost del projecte.

RECURS	QUANTITAT	IMPORT UNITAT	IMPORT TOTAL
Hores Programador	600	60 €	36000 €
Mysql	1	0 €	0 €
Symfony	1	0 €	0 €
Nginx	4	0 €	0 €
Servidors	5	0 €	0 €
		TOTAL	**36000 €**

Figura 1 10 Pressupost projecte

11. TREBALL FUTUR

A continuació, es comenten propostes per a realitzar una segona etapa del projecte:

- **Creació de clients API Rest**

Com s'ha comentat en l'apartat client api en el qual s'estava avaluant el desacoblament de l'arquitectura del sistema, la creació d'API clients facilitaria l'accés per part de l'usuari a les funcionalitats de l'API Rest d'una forma robusta i transparent. Es planteja en una primera etapa implementar API clients en llenguatge PHP i en el llenguatge Javascript.

- **Tests d'integració per testejar els endpoints**

Després de crear tests unitaris per cobrir el nucli del sistema, és a dir, cobrir el codi dels components que formen la capa Domain i la capa Interactor, el següent pas seria testejar els endpoints de l'API Rest. Es testejaria la resposta dels endpoints variant els paràmetres d'entrada. Aquests tests posarien a prova la interacció de tots els elements de totes les capes per a poder generar la resposta del endpoints.

- **Securitzar la comunicació usuari - Api Rest**

Actualment, la comunicació entre l'usuari i l'API Rest es realitza mitjançant el protocol de seguretat SSL [34] i la validació de l'usuari mitjançant l'API key. Un següent pas seria firmar la petició, incloent l'API key, mitjançant una clau privada. La clau seria compartida pel servidor i el client però no seria transmesa durant la comunicació. Aquest procés també el realitzaria el servidor per la seva resposta. D'aquesta forma, es podria validar la integritat de les dades transmeses entre l'usuari i l'API Rest i confirmar que no han sigut modificades durant la comunicació. Aquest punt estaria molt relacionat amb el de creació d'API clients.

- **Canviar el llenguatge de programació de l'API Rest**

Un exercici interessant per demostrar la independencia tecnologica entre l'usuari i l'API Rest seria modificar el llenguatge de programació utilitzat per implementar el sistema. Es proposa el llenguatge de programació python amb el framework Django Rest Framework [75] o el framework Flask Restfull [76].

12. ANNEXOS

12.1. Entorn de desenvolupament

S'ha creat un entorn de desnvolupament utilitzant màquines virtuals les quals es configuren i aprovisionen de forma automàtica. Aquest tipus d'entorn és multiplataforma i conté tots els elements necessaris per a poder executar el sistema i per a poder crear noves funcionalitats.

12.1.1. Vagrant

Vagrant [77] és un programari software el qual permet configurar entorns de desenvolupament virtuals mitjançant un fitxer de configuració VagrantFile. En l'entorn de desenvolupament del projecte, la màquina virtual és un linux de tipus Debian.

12.1.2. Ansible

Ansible [78] és una plataforma software la qual permet l'automatització del aprovisionament de màquines. Aquest fet fa que es pugui automatitzar la configuració de màquines mitjançant fitxers d'automatització en format yml.

12.1.3. Docker

Docker [79] és un projecte software de codi obert que automatitza la creació de contenidors en el qual s'allotgen aplicacions de forma aïllada.

12.1.4. Composer

Composer [80] és un gestor de dependències o llibreries de tercers pel llenguatge PHP. Es poden gestionar les dependències i les versions d'aquestes mitjançant el fitxer de configuració composer.json.

12.1.5. Configuració de l'entorn

Mitjançant aquestes tecnologies i els fitxers de configuració, es crea una màquina virtual de tipus Linux Debian, la qual resideix en la màquina local del programador, la qual és configurada mitjançant els fitxers de configuració ansible. Un cop configurada la màquina virtual i mitjançant la combinació d'ansible i docker, es creen containers individuals de les tecnologies necessàries per a què el sistema pugui funcionar. D'aquesta forma, es crearà un container nginx el qual estarà connectat a un container de tipus php-fpm i aquest últim estarà connectat a un container de tipus mysql i a un altre de tipus redis.

Aquest tipus d'entorn de desenvolupament és molt útil, ja que l'usuari no ha de realitzar configuracions dins de la seva màquina local referent al sistema, sinó que totes les tecnologies i configuracions necessàries per a què el sistema funcioni es troben dins la màquina virtual. Els fitxers de configuració, estan versionats i formen part dels fitxers del projecte.

12.1.6. Manual d'ús

Només la primera, el programador haurà de realitzar els següents passos en la seva màquina local:

- Instal·lar **Vagrant** i les seves dependències
- Instal.lar **Ansible** i les seves dependències
- Obrir un terminal i navegar fins a l'arrel del projecte
- Executar la comanda: **vagrant up**
- Executar la comanda: **php composer.phar install**
- Afegir la línia al fitxer /etc/hosts: 172.21.99.4 badges-io-dev

Es pot veure la configuració de l'entorn i tot el codi del projecte en el següent enllaç: https://bitbucket.org/noLabs/badges-microservice-io/src.

12.2. Documentació API Rest

/api/user/signup

- URL: https://badges-io-dev/api/user/signup
- Verb HTTP: POST
- Format: HAL + JSON
- Paràmetres d'entrada: username, email, password
- Codis d'estat HTTP: 200, 400, 500
- Resposta

```
{
  "id":  text,
  "email": text,
  "username":  text,
   "_links": {
     "login": {
        "href": "https://badges-io-dev/api/user/login"
      },
      "signup": {
        "href": "https://badges-io-dev/api/user/signup"
      },
      "list_badges": {
         "href": "https://badges-io-dev/api/badges/list/{userId}"
      },
      "create_badge": {
        "href": "https://badges-io-dev/api/badge/create"
     }
  }
}
```

/api/user/login

- URL: https://badges-io-dev/api/user/login
- Verb HTTP: PUT
- Format: HAL + JSON
- Paràmetres d'entrada: username ó email, password
- Codis d'estat HTTP: 200, 400, 404, 403, 500
- Resposta

```
{
  "id":  text,
  "email":  text,
  "username": text,
  "_links": {
    "login": {
      "href": "https://badges-io-dev/api/user/login"
    },
    "signup": {
      "href": "https://badges-io-dev/api/user/signup"
    },
    "list_badges": {
      "href": "https://badges-io-dev/api/badges/list/{userId}"
    },
    "create_badge": {
      "href": "https://badges-io-dev/api/badge/create"
    }
  }
}
```

/api/badge/create

- URL: https://badges-io-dev/api/badge/create
- Verb HTTP: POST
- Format: HAL + JSON
- Paràmetres d'entrada: name, description, userId, isMultiUser, imageName, imageWidth, imageHeight, imageFormat, imageFile
- Codis d'estat HTTP: 200, 400, 404, 500
- Resposta

```
{
”id”:  text,
"name":  text,
"description":  text,
"is_multi_user": booleà,
"_links": {
  "self": {
    "href": “https://badges-io-dev/api/badge/{badgeId}”
  },
  "create_badge": {
    "href": “https://badges-io-dev/api/badge/create"
  },
  "delete_badge": {
    "href": “https://badges-io-dev/api/badge/{badgeId}”
  },
  "update_badge": {
    "href": “https://badges-io-dev/api/badge/update”
  },
  "list_badges": {
    "href": “https://badges-io-dev/api/badges/list/{userId}”
  }
},
 "_embedded": {
     "badge:image": {
         "id":  text,
         "name": text,
```

```
        "width": número,
        "height": número,
        ”format": text,
        "href": text
      },
     "badge:owner": {
        "id": “text",
        "email": “text",
        "username": text,
         "_links": {
           "login": {
              "href": “https://badges-io-dev/api/user/login"
            },
            "signup": {
              "href": “https://badges-io-dev/api/user/signup"
             },
            “list_badges": {
              "href": “https://badges-io-dev/api/badges/list/
{userId}”
            },
          ”create_badge": {
            "href": “https://badges-io-dev/api/badge/create
          }
       }
     }
   }
}
```

/api/badge/update

- URL: https://badges-io-dev/api/badge/update
- Verb HTTP: POST
- Format: HAL + JSON
- Paràmetres d'entrada: id, name, description, userId, isMultiUser, imageName, imageWidth, imageHeight, imageFormat, imageFile
- Codis d'estat HTTP: 200, 400, 404, 401, 500
- Resposta

```
 {
     ”id”:  text,
     "name":  text,
     "description": text,
     "is_multi_user": booleà,
     "_links": {
        "self": {
          "href": “https://badges-io-dev/api/badge/{badgeId}”
        },
        "create_badge": {
          "href": “https://badges-io-dev/api/badge/create"
        },
        "delete_badge": {
           "href": “https://badges-io-dev/api/badge/{badgeId}”
        },
        "update_badge": {
           "href": “https://badges-io-dev/api/badge/update”
        },
        “list_badges": {
          "href": “https://badges-io-dev/api/badges/list/{userId}”
        }
      },
"_embedded": {
     "badge:image": {
```

```
  "id":  text,
  "name": text,
  "width": número,
  "height": número,
  ”format": text,
  "href": text
},
"badge:owner": {
  "id": text,
  "email": text,
  "username": text,
    "_links": {
      "login": {
        "href": “https://badges-io-dev/api/user/login"
      },
      “signup": {
        "href": “https://badges-io-dev/api/user/signup"
      },
      "list_badges": {
          "href": “https://badges-io-dev/api/badges/list/{userId}”
       },
       ”create_badge": {
           "href": “https://badges-io-dev/api/badge/create”
        }
      }
    }
  }
}
```

/api/badge/{id}/{userId}

- URL: https://badges-io-dev/api/badge/{badeId}/{userId}
- Verb HTTP: GET
- Format: HAL + JSON
- Paràmetres d'entrada: id, userId
- Codis d'estat HTTP: 200, 400, 404, 401, 500
- Resposta:

```
{
    ”id”:  text,
    "name":  text,
    "description": text,
    "is_multi_user": booleà,
    "_links": {
        "self": {
          "href": “https://badges-io-dev/api/badge/{badgeId}”
        },
        "create_badge": {
          "href": “https://badges-io-dev/api/badge/create"
        },
        "delete_badge": {
          "href": “https://badges-io-dev/api/badge/{badgeId}”
        },
        "update_badge": {
          "href": “https://badges-io-dev/api/badge/update”
        },
        “list_badges": {
          "href": “https://badges-io-dev/api/badges/list/{userId}”
        }
      },
"_embedded": {
      "badge:image": {
          "id":  text,
          "name": text,
```

```
    "width": número,
    "height": número,
    ”format": text,
     "href": text
  },
 "badge:owner": {
     “id": text,
     “email": text,
     "username": text,
    "_links": {
      "login": {
        "href": “https://badges-io-dev/api/user/login"
          },
          “signup": {
            "href": “https://badges-io-dev/api/user/signup"
          },
       "list_badges": {
          "href": “https://badges-io-dev/api/badges/list/
{userId}”
        },
        ”create_badge": {
          "href": “https://badges-io-dev/api/badge/create”
        }
       }
     }
   }
  }
```

/api/badge/list/{userId}

- URL: https://badges-io-dev/api/badge/list/{userId}
- Verb HTTP: GET
- Format: HAL + JSON
- Paràmetres d'entrada: userId
- Codis d'estat HTTP: 200, 400, 404, 500
- Resposta:

```
[{
      ”id”:  text,
      "name":  text,
      "description": text,
      "is_multi_user": booleà,
      "_links": {
         "self": {
           "href": “https://badges-io-dev/api/badge/{badgeId}”
         },
         "create_badge": {
           "href": “https://badges-io-dev/api/badge/create"
         },
         "delete_badge": {
            "href": “https://badges-io-dev/api/badge/{badgeId}”
         },
         "update_badge": {
            "href": “https://badges-io-dev/api/badge/update”
         },
         “list_badges": {
            "href": “https://badges-io-dev/api/badges/list/{userId}”
         }
       },
 "_embedded": {
       "badge:image": {
            "id":  text,
            "name": text,
          "width": número,
```

```
    "height": número,
    ”format": text,
     "href": text
 },
"badge:owner": {
     “id": text,
     “email": text,
     "username": text,
    "_links": {
      "login": {
        "href": “https://badges-io-dev/api/user/login"
          },
          “signup": {
            "href": “https://badges-io-dev/api/user/signup"
          },
      "list_badges": {
          "href": “https://badges-io-dev/api/badges/list/
{userId}”
       },
       ”create_badge": {
           "href": “https://badges-io-dev/api/badge/create”
       }
      }
     }
   }
 }]
```

/api/badge/{id}/{userId}

- URL: https://badges-io-dev/api/badge/{badeId}/{userId}
- Verb HTTP: DELECTE
- Format: HAL + JSON
- Paràmetres d'entrada: id, userId
- Codis d'estat HTTP: 200, 400, 404, 401, 500
- Resposta:

```
{
    “status": text,
    “message": text,
    "_links": {
      "create_badge": {
        "href": "https://badges-io-dev/api/badge/create"
      },
     "list_badges": {
        "href": “https://badges-io-dev/api/badges/list/{userId}
     },
     "login": {
        "href": "https://badges-io-dev/api/user/login"
     },
     "signup": {
        "href": "https://badges-io-dev/api/user/signup"
    }
}
```

Bibliografia

[1] Web Atrapalo
[2] Informació empresa Atrapalo
[3] Responsive Web Design
[4] Gamification: Toward a Definition
[5] La Gamificación como participante en el desarrollo del B-learning
[6] Web American Airlines
[7] Programa AAdvantage
[8] Programa Viajero Frecuente
[9] Web Stackoverflow
[10] Informació Empresa Stackoverflow
[11] Stackoverflow Badges
[12] Stackoverflow Badge Question Estelar
[13] Stackoverflow Badge Guru
[14] Web Foursquare
[15] Informació Empresa Foursquare
[16] Foursquare Badges
[17] Moodle Badge
[18] YITH WooCommerce Badge Management
[19] Open Badges
[20] Service-Oriented Architecture (SOA) Definition
[21] Microservices Definition
[22] Informació Microservices
[23] XML-RPC Definition
[24] SOAP (Simple Object Access Protocol) definition
[25] Conceptos sobre APIs REST
[26] SOLID principles
[27] Definició Unit Test
[28] Definició Command Pattern
[29] Alistair Cockburn: Ports And Adapters Architecture
[30] Garfixia Software Architectures: Ports And Adapters / Hexagonal Architecture
[31] Robert C. Martin: The Clean Architecture
[32] Jeffrey Palermo: The Onion Architecture
[33] Martin Fowler: Inversion of control containers and the dependency injection pattern
[34] Evans, Eric (2004) Domain-Driven Design: Trackling Complexity in the Heart of Software. Addison-Wesley. ISBN 978-032-112521-7
[35] Unit Test
[36] PHPUnit
[37] Integration Test
[38] Code Coverage
[39] Business Logic
[40] TDD
[41] "Extreme Programming". Computerworld. Retrieved January 11, 2011.

[42] Beck, K. Test-Driven Development by Example, Addison Wesley - Vaseem, 2003 ee Copeland (December 2001)
[43] Robert C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship 2009
[44] MySQL
[45] Doctrine
[46] Data Transfer Object
[47] Nginx
[48] Symfony
[49] Bundle
[50] Service Container
[51] REST
[52] HTTP
[53] HTTP Status Code Definition
[54] Richardson Maturity Model
[55] The Maturity Heuristic
[56] HTTP Verbs
[57] HATEOAS
[58] JSON
[59] HAL
[60] FOSRestBundle
[61] JMSSerializer
[62] NelmioApiDocBundle
[63] Top Six Reasons to Use API Keys
[64] Annotations
[65] HTML
[66] Twig
[67] Bootstrap
[68] Console Commands
[69] Redis
[70] Predis
[71] Amazon Web Services
[72] Google Cloud Computing
[73] High Availability
[74] SSL
[75] Django Rest Framework
[76] Flask Restfull
[77] Vagrant
[78] Ansible
[79] Docker
[80] Composer